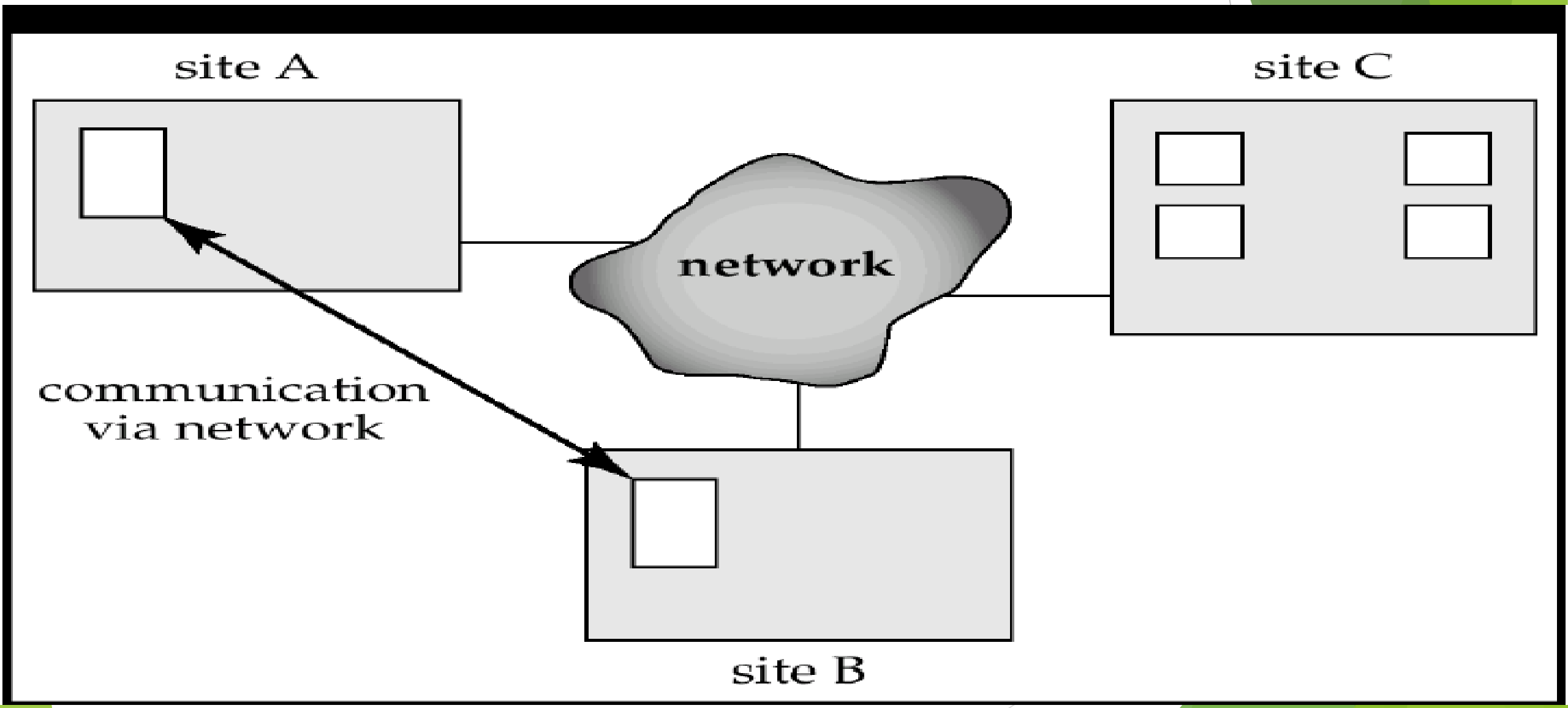
Module 5

Distributed Database Management systems

- A distributed database can be defined as a logically interrelated collection of shared data physically distributed over a computer network.
- Distributed DBMS can be defined as the software system that permits the management of the distributed database and makes the distribution transparent to users.
- Users access the distributed database via applications. Applications can be classified as local applications (that do not require data from other sites) and global applications (that requires data from other sites).

DDBMS Architecture



DDBMS Architecture

- In a **distributed database system, the database is stored on several computers.**
- **The** computers in a distributed system communicate with one another through various communication media, such as high-speed networks or telephone lines.
- They do not share main memory or disks. The computers in a distributed system may vary in size and function, ranging from workstations up to mainframe systems.
- The computers in a distributed system are referred to by a number of different names, such as sites or nodes, depending on the context in which they are mentioned.
- We mainly use the term site, to emphasize the physical distribution of these systems

Advantages of Distributed Database Management systems

Advantages of DDBMS

a) Capacity and incremental growth

b) Reliability and availability

c) Efficiency and flexibility

d) Sharing

Advantages of Distributed Database Management systems

a) Capacity and growth

- As the organisation grows, new sites can be added easily

b) Reliability and availability

- Even when a portion of a system (i.e. a local site) is down, the overall system remains available.
- With replicated data, the failure of one site still allows access to the replicated copy of the data from another site. The remaining sites continue to function.
- The greater accessibility enhances the reliability of the system

Advantages of Distributed Database Management systems

c) Efficiency and flexibility

- Data is physically stored close to the anticipated point of use. Hence if usage patterns change then data can be dynamically moved or replicated to where it is most needed.

d) Distributed database sharing

- An advantage of distributed databases is that users at a given site are able to access data stored at other sites and at the same time retain control over the data at their own site.

Disadvantages of Distributed Database Management systems

a) Complexity. A distributed system, which hides its distributed nature from the end user, is more complex than the centralised system.

b) Cost : Increased complexity means that the acquisition and maintenance costs of the system are higher than those for a centralised DBMS.

- Requires additional hardware to establish a network between sites.

c) Security : In a centralized system, access to the data can be easily controlled. But in DDBMS, network, replicated data all have to be made secure and is very difficult

d) Lack of standards : No tools or methodologies to help users to convert a centralized DBMS into a DDBMS.

Types of DDBMS

Homogeneous distributed database

- All sites have identical software
- Are aware of each other and agree to cooperate in processing user requests.
- Each site surrenders part of its autonomy in terms of right to change schemas or software
- Appears to user as a single system

Heterogeneous distributed database

- Different sites may use different schemas and software
- Difference in schema is a major problem for query processing
- Difference in software is a major problem for transaction processing
- Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing

Distributed Data Storage

- Consider a relation *r that is to be stored in the database. There are two approaches* to storing th is relation in the distributed database:
- **Replication. The system maintains several identical replicas (copies) of the** relation, and stores each replica at a different site. The alternative to replication is to store only one copy of relation *r.*
- **Fragmentation. The system partitions the relation into several fragments,** and stores each fragment at a different site.

Data Replication

- If relation r is replicated, a copy of relation r is stored in two or more sites. In the most extreme case, we have **full replication, in which a copy is stored in every** site in the system

Advantages

- **Availability. If one of the sites containing relation r fails, then the relation r** can be found in another site. Thus, the system can continue to process queries involving r, despite the failure of one site.
- **Increased parallelism. In the case where the majority of accesses to the relation** r result in only the reading of the relation, then several sites can process queries involving r in parallel.
- The more replicas of r there are, the greater the chance that the needed data will be found in the site where the transaction is executing. Hence, data replication minimizes movement of data between sites.

Data Replication

Drawbacks

- **Increased overhead on update. The system must ensure that all replicas of a** relation r are consistent; otherwise, erroneous computations may result.
- Thus, whenever r is updated, the update must be propagated to all sites containing replicas.
- The result is increased overhead. For example, in a banking system, where account information is replicated in various sites, it is necessary to ensure that the balance in a particular account agrees in all sites
- **Increased complexity of concurrency contro**l: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.

Data Fragmentation

- Fragmentation consists of breaking a relation into smaller relations or fragments, and storing the fragments, possibly at different sites.
- If relation r is fragmented, r is divided into a number of fragments r1, r2, . . . , rn.
- These fragments contain sufficient information to allow reconstruction of the original relation r.
- A table may be broken up horizontally, vertically, or a combination of both.
- Three types of fragmentation:
- Horizontal fragmentation
- Vertical fragmentation
- Mixed fragmentation

Data Fragmentation

- **Horizontal fragmentation refers to the division of a relation into subsets (fragments) of tuples (rows).**
- **Each** fragment is stored at a different node, and each fragment has unique rows. However, the unique rows all have the same attributes (columns). In short, each fragment represents the equivalent of a SELECT statement, with the WHERE clause on a single attribute.
- The union of the horizontal fragments must be equal to the original relation

EID	NAME	Gender	CITY	SALARY
5364	Jones	M	Madras	3500
5365	Alice	F	Chicago	3200
5366	Smith	M	Chicago	4800
5368	Jaya	F	Pune	5000

EID	NAME	Gender	CITY	SALARY
5364	Jones	M	Madras	3500
5366	Smith	M	Chicago	4800

EID	NAME	Gender	CITY	SALARY
5365	Alice	F	Chicago	3200
5368	Jaya	F	Pune	5000

Data Fragmentation

- In the given example,we are dividing fragment based on some condition such that all data with gender male will reside at one fragement and others at different fragment
- Vertical fragmentation refers to the division of a relation into attribute (column) subsets.
- Each subset (fragment) is stored at a different node, and each fragment has unique columns—with the exception of the key column, which is common to all fragments. This is the equivalent of the PROJECT statement in Relational Algebra

Cust_id	CustName	City	Gender
1	Boby	Mumbai	Male
2	Alice	Bangalore	Female
3	John	Agra	Male
4	Jaya	Pune	Female

Data Fragmentation

- In the given example ,we are storing 2 columns at one fragment and 3 columns at another fragment

Cust_id	CustName
1	Boby
2	Alice
3	John
4	Jaya

Cust_id	City	Gender
1	Mumbai	Male
2	Bangalore	Female
3	Agra	Male
4	Pune	Female

Data Fragmentation

- Mixed fragmentation refers to a combination of horizontal and vertical strategies.
- **In other words, a table** may be divided into several horizontal subsets (rows), each one having a subset of the attributes (columns).
- Mixed Fragmentation can be done as to create a set or group of horizontal fragments and then create vertical fragments one or more of the horizontal fragments

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000

$$\pi_{name}\,(\sigma_{dept\ name} = \text{“}Physics\text{”}\ (instructor))$$

Einstein
Gold

Distributed Transactions

- Transaction may access data at several sites.
- A local transaction is one that accesses data only from sites where the transaction was initiated.
- A **global transaction, on the other hand, is one that either accesses data in a site** different from the one at which the transaction was initiated, or accesses data in several different sites

a)System Structure

- Each site has its own local transaction manager, whose function is to ensure the ACID properties of those transactions that execute at that site. The various transaction managers cooperate to execute global transactions

Distributed Transactions

- Each site contains two subsystems:
- The **transaction manager manages the execution of those transactions (or** sub transactions) that access data stored in a local site. Note that each such transaction may be either a local transaction (that is, a transaction that executes at only that site) or part of a global transaction (that is, a transaction that executes at several sites).
- The **transaction coordinator coordinates the execution of the various transactions** (both local and global) initiated at that site.

Distributed Transactions

- Each site has a local transaction manager responsible for:
- Maintaining a log for recovery purposes
- Participating in coordinating the concurrent execution of the transactions executing at that site.
- Each site has a transaction coordinator, which is responsible for:
- Starting the execution of the transaction
- Breaking the transaction into a number of sub transactions and distributing these sub transactions to the appropriate sites for execution
- Coordinating the termination of the transaction, which may result in the transaction being committed at all sites or aborted at all sites

Distributed Transactions

b)System Failure Modes

- A distributed system may suffer from the same types of failure that a centralized system does (for example, software errors, hardware errors, or disk crashes).
- There are, however, additional types of failure with which we need to deal in a distributed environment. The basic failure types are:
- Failure of a site.
- Loss of messages. Handled by network transmission control protocols such as TCP-IP
- Failure of a communication link. Handled by network protocols, by routing messages via alternative links
- Network partition:A network is said to be **partitioned when it has been split into two** or more subsystems that lack any connection between them

Distributed Transactions

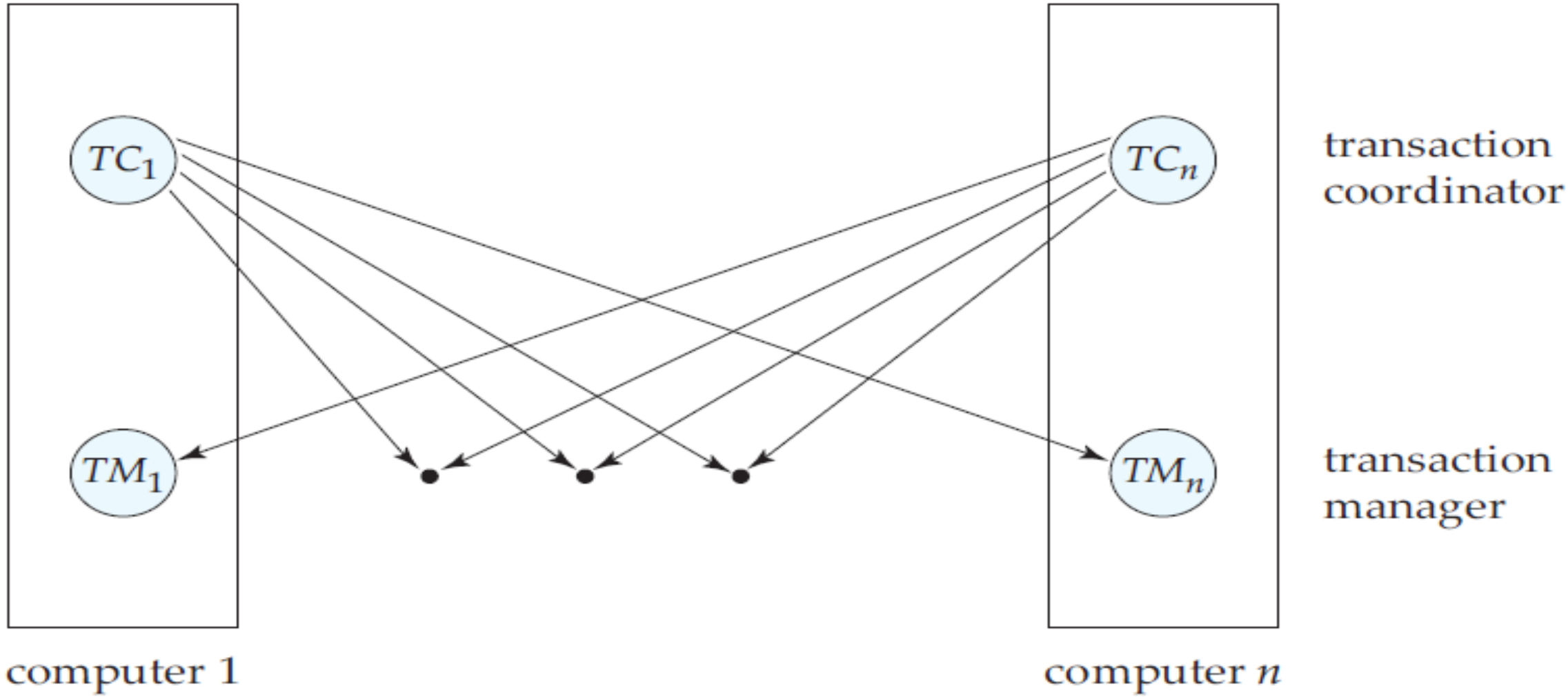
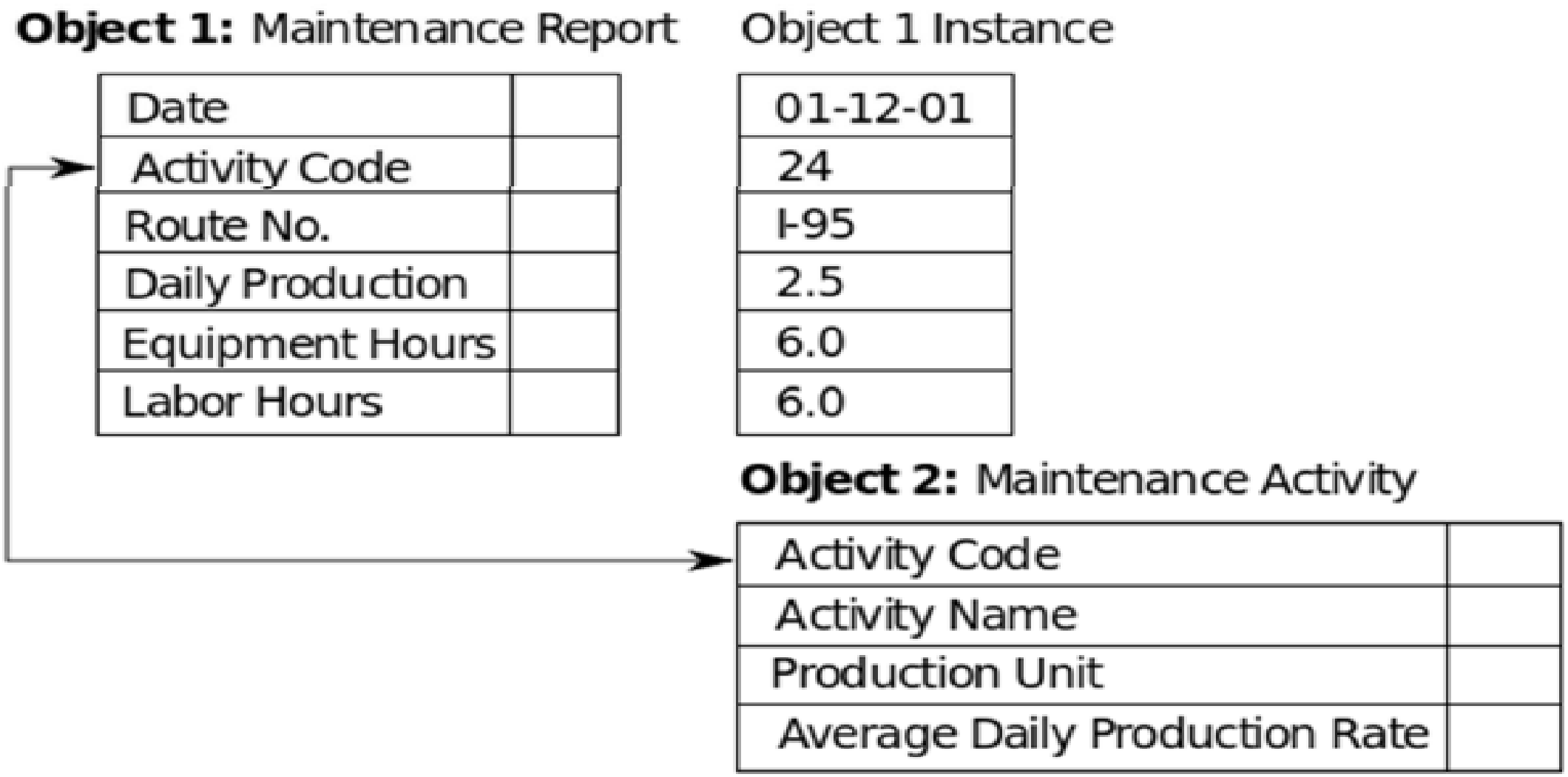


Figure 19.2 System architecture.

Object database

- An object database is a database management system in which information is represented in the form of objects as used in object-oriented programming.
- Object databases are different from relational databases which are table-oriented.
- Main feature of object oriented database is that an object continues to exist once the program that created it has finished running is known as persistence

Object-Oriented Model



Object Database

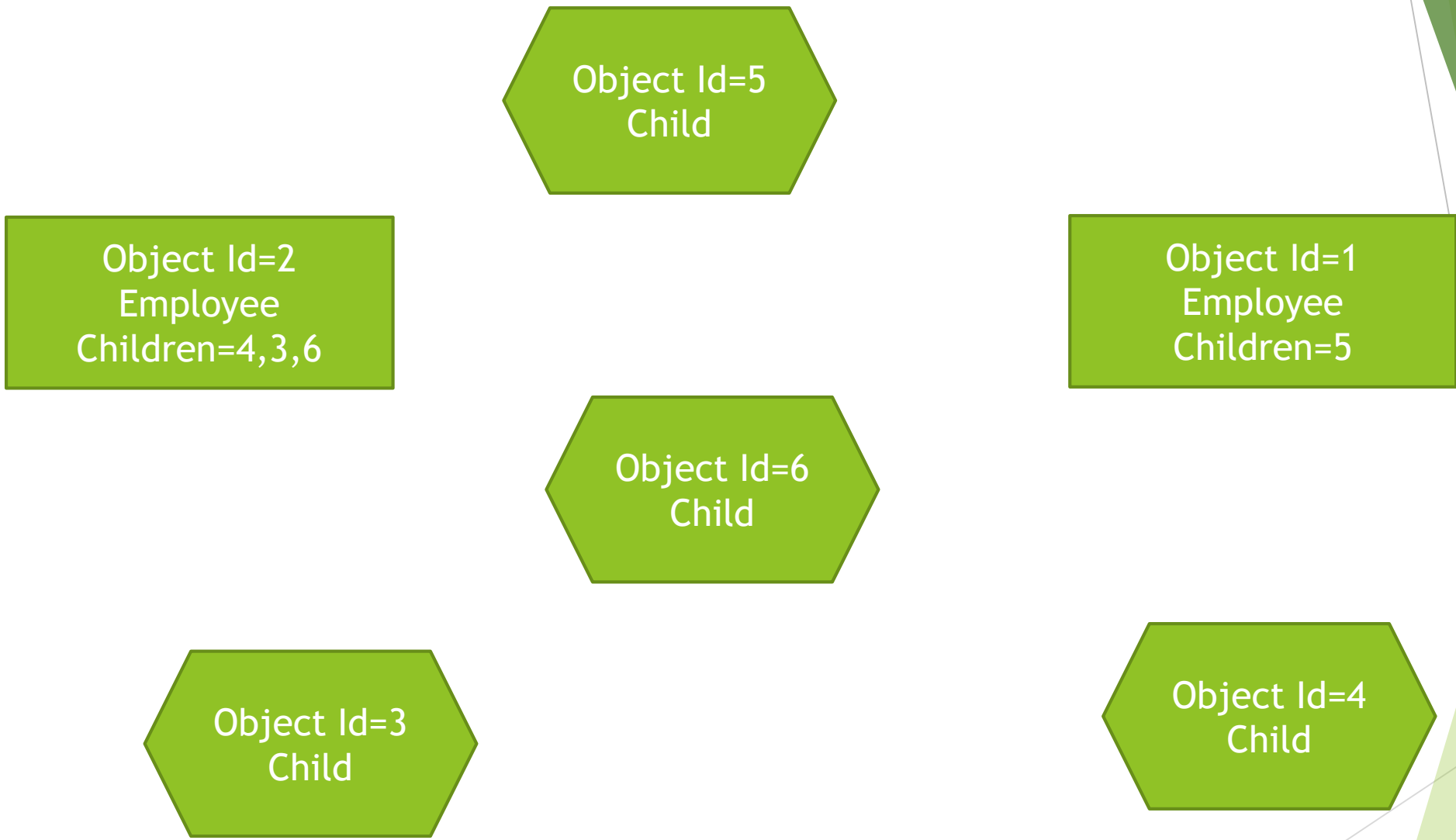
OO Identifiers

- A relational db represents data relationships by having matching primary key-foreign key data.
- No data structures within the db that forms links between the tables; the relationships are used as needed by joining tables.
- A pure OO db represents relationships by including object identifiers within an object to indicate other objects to which its related.
- OO identifier is an internal db identifier for each individual object
- Users, whether programmers or end users working within an interactive query tool, never see or manipulate these identifiers
- OO identifiers, they are assigned and only used by the DBMS.
- The meaning of the identifier is unique to each DBMS.

Object Datbase

OO Identifiers

- It may be an arbitrary value or it may include the information necessary to locate the object in a database file.
- For example, an object identifier might include the *page number and the offset from the beginning of the page for the file in which the object is stored.*
- For example, we were relating objects of an Employee class to objects of a Child class.
- Each employee object would be related to one or more child objects.
- Each rectangle in the diagram is an employee object and each hexagon is a child object, all of which have their own unique object identifiers.


Object Id=5
Child
Object Id=2
Employee
Children=4,3,6
Object Id=1
Employee
Children=5
Object Id=6
Child
Object Id=3
Child
Object Id=4
Child

Object Database

- The two employee objects have attributes named Children.
- The values of those attributes are the object identifiers of the employee's offspring.
- There are *two important aspects to this method of representing data relationships:*
- For this mechanism to work, an object's identifier must not change while the object is a part of the database.
- If it were to change, the DBMS would be unable to locate related objects because the identifiers would not indicate the correct related objects.
- The only relationships that can be used for querying or traversing the database are those that have been pre defined by storing the object identifiers of related objects in attributes.

Object Relational Data Model

- **Object-relational database systems, that is, database systems based** on the object-relation model, provide a convenient migration path for users of relational databases who wish to use object-oriented features.
- The **object-relational data model extends the relational data model** by providing a richer type system including complex data types and object orientation.
- Relational ----object relational database ------object oriented database systems.

Object Relational Data Model

Complex Types :

- Any data that does not fall into the traditional field structure (alpha, numeric, dates) of a relational DBMS are complex data types.
- Examples of complex data types are bills of materials, word processing documents, maps, time-series, images and video.
- With complex type systems and object orientation, we can represent E-R model concepts, such as identity of entities, multi valued attributes, and generalization and specialization directly, without a complex translation to the relational model.

Extensions to SQL to support complex types include:

- Collection and large object types
- Structured types
- Inheritance
- Object orientation

Object Relational Data Model

Complex types include Collection and Large Object Types.

Collection Types

- Table definition differs from table definitions in ordinary relational databases,
- Since it allows attributes that are sets, thereby permitting multi valued attributes of E-R diagrams to be represented directly.
- Sets, arrays, multi sets are an instance of collection types.

Arrays

- E.g.Author-array varchar (20) array[10]
- Can access elements of array in usual fashion:
- E.g. author-array[1]

Multi sets

- ie, unordered collections, where an element may occur multiple times.

Object Relational Data Model

Complex types include Collection and Large Object Types.

Collection types.....

- ARRAY : ordered 1D array with maximum number of elements without duplicates .
- LIST : ordered collection that allows duplicates.
- SET : unordered collection without duplicates .
- MULTISET : unordered collection that allows duplicates.

Object Relational Data Model

Complex types include Collection and Large Object Types.

Large-object data types

- Many current-generation database applications need to store attributes that can be large (of the order of many kilobytes), such as a photograph of a person, or very large (of the order of many megabytes or even gigabytes), such as a high resolution medical image or video clip.
- Hence for character data (**clob) and binary data (blob) and structured types are** also supported.
- The letters “lob” in these data types stand for “Large Object”. For example, we may declare attributes

book-review clob(10KB)

image blob(10MB)

movie blob(2GB))

Object Relational Data Model

Structred Types

- Structured types allow composite attributes of E-R diagrams to be represented directly.

Eg 1 **create type Name as**

(firstname **varchar(20),**

lastname **varchar(20))**

final;

Eg 2: **create type Address as**

(street **varchar(20),**

city **varchar(20),**

zipcode **varchar(9))**

not final;

- Such types are called **user-defined types**

Object Relational Data Model

- We can use these types to create composite attributes in a relation, by simply declaring an attribute to be of one of these types.
- For example, we could create a table person as follows:

create table person (name Name,

address Address,

dateOfBirth **date);**

- The components of a composite attribute can be accessed using a **“dot” notation;**for instance **name.firstname returns the firstname component of the name** attribute.
- An access to attribute name would return a value of the structured type Name.

Object Relational Data Model

- We can also **create a table whose rows are of a user-defined type.**
- For example, we could define a type PersonType and create the table person as follows:

create type PersonType as (

name Name,

address Address,

dateOfBirth **date)**

not final

create table person of PersonType;

Object Relational Data Model

- An alternative way : **use unnamed row types.**
- For instance, the relation representing person information could have been created using row types as follows:

create table person r (

name **row (firstname varchar(20),**

lastname **varchar(20)),**

address **row (street varchar(20),**

city **varchar(20),**

zipcode **varchar(9)),**

dateOfBirth **date);**

The query finds the last name and city of each person.

Select name.lastname, address.city from person

Object Relational Data Model

- A structured type can have **methods defined on it.**

create type PersonType as (

name Name,

address Address,

dateOfBirth **date)**

not final

method ageOnDate(onDate date)

returns interval year;

- We create the method body separately:

create instance method ageOnDate
(onDate date)

returns interval year

for PersonType

begin

return onDate − self.dateOfBirth;

end;

Object Relational Data Model

Type inheritance

- Type inheritance applies to **named row types only.**
- You can use inheritance to group named row types into a type hierarchy in which each subtype inherits the representation (data fields) and the behavior of the super type under which it is defined.
- The keyword **final says that subtypes may not be created from the given type,** while **not final says that subtypes may be created.**

Object Relational Data Model

create type Person

(name **varchar(20),**

address **varchar(20));**

- we can use inheritance to define the student and teacher types in SQL :

create type Student

under Person

(degree **varchar(20),**

department **varchar(20));**

create type Teacher

under Person

(salary **integer,**

department **varchar(20));**

Object Relational Data Model

- For **multiple inheritance, we can define a type for teaching assistant as** follows:

create type TeachingAssistant under Student, Teacher;

- To avoid a conflict between the two occurrences of department, we can rename them by using an as clause, as in this definition of the type

create type TeachingAssistant under Student with (department as student dept),Teacher **with** (department as teacher dept);

Object Relational Data Model

Table Inheritance

- Suppose we have a table create table people of Person;
- We can then define tables students and teachers as subtables of people, as follows:

create table students of Student under people;

create table teachers of Teacher under people;

- The types of the subtables (Student and Teacher in the above example) are subtypes of the type of the parent table (Person in the above example).
- As a result, every attribute present in the table people is also present in the subtables students and teachers.
- In this way every tuple present in students or teachers becomes implicitly present in people

Object Relational Data Model

Table Inheritance

- Thus, if a query uses the table people, it will find not only tuples directly inserted into that table, but also tuples inserted into its subtables, namely students and teachers but can use only those attributes that are present in people can be accessed by that query.
- Keyword only : permits us to find tuples that are in people but not in its subtables.

Object Relational Data Model

Object-Identity and Reference Types

- Object-oriented languages provide the ability to refer to objects.
- An attribute of a type can be a reference to an object of a specified type.
- Eg : Type Department with a field name and a field head that is a reference to the type Person, and a table departments of type Department, as follows:

```
create type Department (
                        name  varchar(20),
                        head ref(Person) scope people);
create table departments of Department;
```

XML

- XML is not a replacement for HTML
- XML is a meta language for describing mark-up languages.
- It provides a facility to define tags and the structural relationship between them
- *XML means "eXtensible Markup Language"*
- *extensible - no fixed format like HTML*
- XML is a very simple and universal way of storing and transferring any textual kind
- XML does not predefine any tags
- *Specifies the structure and content of document*

EXAMPLE XML DOCUMENT

```
<?xml version="1.0"?>
<CSE>
   <STUDENT>
         <REGNO>234249</REGNO>
         <NAME>Abraham</NAME>
         <COURSE>BTech</COURSE>
         <SEM>06</SEM>
   </STUDENT>
   <STUDENT>
         <REGNO>464652</REGNO>
         <NAME>Abu</NAME>
         <COURSE>BTech</COURSE>
         <SEM>08</SEM>
   </STUDENT>
</CSE>
```

XML

- XML stands for Extensible Markup Language.
- XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data.
- XML provides a way to represent data that have nested structure, and furthermore allows a great deal of flexibility in structuring of data, which is used in business applications.
- It is particularly useful as a data format when an application must communicate with another application, or integrate information from several other applications.

<?xml version="1.0"?>

<contact-info>

<company>

TutorialsPoint </company>

<contact-info>

XML

XML usage

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

XML

Root Element: An XML document can have only one root element.

- For example, following is not a correct XML document, because both the x and y elements occur at the top level without a root element:

<x>...</x>

<y>...</y>

- The following example shows a correctly formed XML document:

<root>

<x>...</x>

<y>...</y>

</root>

- **Case Sensitivity : The names of XML-elements are case-sensitive. That means** the name of the start and the end elements need to be exactly in the same case.
- For example, <contact-info> is different from <Contact-Info>.

XML

Mark Up

- XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable.
- A markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.
- In XML the markup takes the form of tags enclosed in angle-brackets, <>.
- Tags are used in pairs, with <tag> and </tag> delimiting the beginning and the end of the portion of the document to which the tag refers

XML

- For example, the title of a document might be marked up as follows.

<title>Database System Concepts</title>

<message>

<text>Hello, world!</text>

</message>

- The tags <message> and </message> mark the start and the end of the XML code fragment.
- The tags <text> and </text> surround the text Hello, world!.

XML

- Though an XML representation has significant advantages when it is used to exchange data
- The presence of the tags makes the message self-documenting;
- The format of the document is not rigid.
- A wide variety of tools are available to assist in its processing
- Just as SQL is the dominant language for querying relational data, XML is becoming the dominant format for data exchange.

Structure of XML Data

- The fundamental construct in an XML document is the element.
- An element is simply a pair of matching start- and end-tags, and all the text that appears between them.
- XML documents must have a single root element that encompasses all other elements in the document.

XML

Tags and Elements

- XML-elements is also called XML-nodes or XML-tags.
- The names of XML-elements are enclosed in triangular brackets < >
- Eg: <element>

Syntax Rules for Tags and Elements

- Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

<element>....</element>

or in simple-cases, just this way:

<element/>

- Nesting of Elements: An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap.

<bank-1>
<customer>
<customer-name> Hayes </customer-name>
<customer-street> Main </customer-street>
<customer-city> Harrison </customer-city>
<account>
<account-number> A-102 </account-number>
<branch-name> Perryridge </branch-name>
<balance> 400 </balance>
</account>
<account>
…
</account>
</customer>
.
.
</bank-1>

HTML	XML
HTML is an abbreviation for HyperText Markup Language.	XML stands for eXtensible Markup Language.
HTML was designed to display data with focus on how data looks.	XML was designed to be a software and hardware independent tool used to transport and store data, with focus on what data is.
HTML is a markup language itself.	XML provides a framework for defining markup languages.
HTML is a presentation language.	XML is neither a programming language nor a presentation language.
HTML is case insensitive.	XML is case sensitive.
HTML is used for designing a web-page to be rendered on the client side.	XML is used basically to transport data between the application and the database.
HTML has it own predefined tags.	While what makes XML flexible is that custom tags can be defined and the tags are invented by the author of the XML document.
HTML is not strict if the user does not use the closing tags.	XML makes it mandatory for the user the close each tag that has been used.
HTML does not preserve white space.	XML preserves white space.
HTML is about displaying data,hence static.	XML is about carrying information,hence dynamic.

XML Document Schema

- Language for expressing constraints about XML document. It describes the structure of an XML document.

1 DTD

- The type of an XML document can be specified using a DTD

DTD constraints structure of XML data

- What elements can occur
- What attributes can/must an element have
- What subelements can/must occur inside each element, and how many times.

- DTD does not constrain data types

- All values represented as strings in XML

DTD syntax

- <!ELEMENT element (subelements-specification) >
- <!ATTLIST element (attributes) >

XML Document Schema

- Subelements can be specified as
- names of elements, or
- #PCDATA (parsed character data), i.e., character strings
- EMPTY (no subelements) or ANY (anything can be a subelement)

Example

- <! ELEMENT depositor (customer-name account-number)>
- <! ELEMENT customer-name (#PCDATA)>
- <! ELEMENT account-number (#PCDATA)>
- Subelement specification may have regular expressions
- <!ELEMENT bank ((account | customer | depositor)+)>

● Notation:

- – "|" - alternatives , "+" - 1 or more occurrences

Bank DTD

```
<!DOCTYPE bank [
<!ELEMENT bank ( ( account | customer |
depositor)+)>
<!ELEMENT account (account-number branch-name balance)>
<! ELEMENT customer(customer-name customer-street customer-city)>
<! ELEMENT depositor (customer-name account-number)>
<! ELEMENT account-number (#PCDATA)>
<! ELEMENT branch-name (#PCDATA)>
<! ELEMENT balance(#PCDATA)>
<! ELEMENT customer-name(#PCDATA)>
<! ELEMENT customer-street(#PCDATA)>
] >
```

DTD

- The allowable attributes for each element are declared in the DTD
- Attributes may be specified to be of type CDATA, ID, IDREF, or IDREFS; the type CDATA simply says that the attribute contains character data,
- For instance, the following line from a DTD specifies that element course has an attribute of type course id, and a value must be present for this attribute:
- <!ATTLIST course course id CDATA #REQUIRED>
- Attributes must have a type declaration and a default declaration.
- The default declaration can consist of a default value for the attribute or #REQUIRED, meaning that a value must be specified for the attribute in each element, or #IMPLIED, meaning that no default value has been provided, and the document may omit this attribute.
- If an attribute has a default value, for every element that does not specify a value for the attribute, the default value is filled in automatically when the XML document is read.

DTD

- An attribute of type ID provides a unique identifier for the element; a value that occurs in an ID attribute of an element must not occur in any other element in the same document. At most one attribute of an element is permitted to be of type ID
- An attribute of type IDREF is a reference to an element; the attribute must contain a value that appears in the ID attribute of some element in the document.

DTD with ID and IDREFS attribute types.

<!DOCTYPE university-3 [

<!ELEMENT university (
(department|course|instructor)+)>

<!ELEMENT department (building, budget)>

<!ATTLIST department

dept name ID #REQUIRED >

<!ELEMENT course (title, credits)>

<!ATTLIST course

course id ID #REQUIRED

dept name IDREF #REQUIRED

instructors IDREFS #IMPLIED >

<!ELEMENT instructor (name, salary)>

<!ATTLIST instructor

IID ID #REQUIRED >

- dept name IDREF #REQUIRED >
- · · · declarations for title, credits, building,
- budget, name and salary · · ·
-] >

DTD with ID and IDREFS attribute types.

<university-3>

<department dept name="Comp. Sci.">

<building> Taylor </building>

<budget> 100000 </budget>

</department>

<department dept name="Biology">

<building> Watson </building>

<budget> 90000 </budget>

</department>

<course course id="CS-101" dept name="Comp.Sci" instructors="10101 83821">

<title> Intro. to Computer Science </title>

<credits> 4 </credits>

</course>

<course course id="BIO-301" dept name="Biology"

instructors="76766">

<title> Genetics </title>

DTD with ID and IDREFS attribute types.

<credits> 4 </credits>

</course>

<instructor IID="10101" dept name="Comp. Sci.">

<name> Srinivasan </name>

<salary> 65000 </salary>

</instructor>

<instructor IID="83821" dept name="Comp. Sci.">

<name> Brandt </name>

<salary> 72000 </salary>

</instructor>

<instructor IID="76766" dept name="Biology">

<name> Crick </name>

<salary> 72000 </salary>

</instructor>

</university-3>

Limitations of DTD

- No typing of text elements and attributes
- All values are strings, no integers, reals, etc.
- Difficult to specify unordered sets of sub elements
- Order is usually irrelevant in databases
- (A | B)* allows specification of an unordered set, but
- Cannot ensure that each of A and B occurs only once
- IDs and IDREFs are untyped
- The owners attribute of an account may contain a reference to another account, which is meaningless owners attribute

NameSpace

- XML data has to be exchanged between organizations
- Same tag name may have different meaning in different organizations, causing confusion on exchanged documents
- Specifying a unique string as an element name avoids confusion
- Better solution: use unique- name: element-name
- Avoid using long unique names all over document by using XML Namespaces
- <bank Xmlns:FB='http://www.FirstBank.com'>
- <FB:branch>
- <FB:branchname>Downtown</FB:branchname>
- <FB:branchcity> Brooklyn </FB:branchcity>
- </FB:branch>
- </bank>

XML Document Schema

2. XML Schema

- The XML Schema language is also referred to as XML Schema Definition (XSD).

Among the benefits that **XML Schema offers over DTDs are these:**

- It allows **user-defined types to be created.**
- It allows the text that appears in elements to be **constrained to specific** types, such as numeric types in specific formats or even more complicated types such as lists or union.
- It allows types to be restricted to create **specialized types, for instance by** specifying minimum and maximum values.

XML Document Schema

2. XML Schema

- XML Schema is a more sophisticated schema language which addresses the drawbacks of DTDs.

Supports

- Typing of values

E.g. integer, string, etc

Also, constraints on min/max values

- User defined types
- Is itself specified in XML syntax, unlike DTDs
- Many more features

List types, uniqueness and foreign key constraints, inheritance ..

- Significantly more complicated than DTDs, not yet widely used

XML Document Schema

2. XML Schema

- It allows complex types to be extended by using a form of **inheritance.**
- It is a superset of DTDs.
- It allows **uniqueness and foreign key constraints.**
- It is integrated with namespaces to allow different parts of a document to conform to different schema.
- It is itself specified by XML syntax.

XML Schema version of Bank DTD

- <xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
- <xsd:element name="bank" type="BankType"/>
- <xsd:element name="account">
- <xsd:complexType>
- <xsd:sequence>
- <xsd:element name="account-number"type="xsd:string"/>
- <xsd:element name="branch-name" type="xsd:string"/>
- <xsd:element name="balance" type="xsd:decimal"/>
- </xsd:squence></xsd:complexType>

<xsd:complexType name="BankType">

<xsd:squence>

<xsd: element ref="account" minoccurs="0" maxoccurs="unbounded"/>

<xsd: element ref="customer" minoccurs="0" maxoccurs="unbounded"/>

<xsd: element ref="depositor" minoccurs="0" maxoccurs="unbounded"/>

Storage of XML data

1.Store as string. A simple way to store XML data in a relational database is to store each child element of the top-level element as a string in a separate tuple in the database.

- The XML data for bank could be stored as a set of tuples in a relation elements(data), with the attribute data of each tuple storing one XML element (account, customer, or depositor) in string form.

2.Tree representation. Arbitrary XML data can be modeled as a tree and stored using a pair of relations:

- nodes(id, type, label, value)
- child(child-id, parent-id)
- Each element and attribute in the XML data is given a unique identifier. A tuple inserted in the nodes relation for each element and attribute with its identifier (id), its type (attribute or element), the name of the element or attribute (label), and the text value of the element or attribute (value).

Storage of XML data

3 **Map to relations. In this approach, XML elements whose schema is** known are mapped to relations and attributes. Elements whose schema is unknown are stored as strings, or as a tree representation

Distributed database patterns

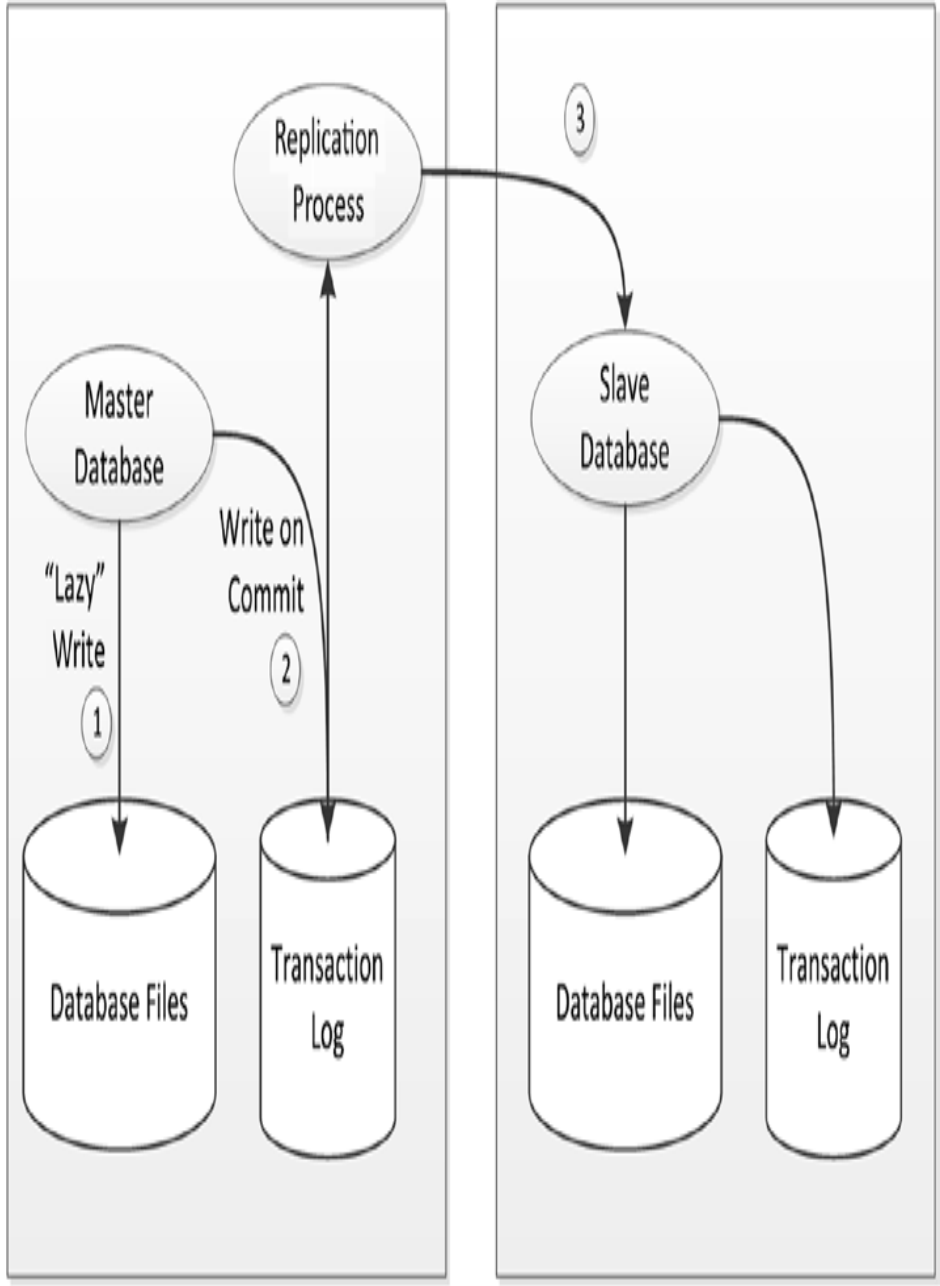
I Distributed Relational Databases

1.1 Replication

- Database replication was initially adopted as a means of achieving high availability.
- Using replication, database administrators could configure a *standby database that could take over for the primary database in* the event of failure
- Database replication often took advantage of the *transaction log that most relational databases used* to support ACID transactions

Replication

- Figure illustrates the log-based replication approach. Database transactions are written in an asynchronous "lazy" manner to the database files (1), but a database transaction immediately writes to the transaction log upon commit (2).
- The replication process monitors the transaction log and applies transactions as they are written to the read-only slave database (3).
- Replication is usually asynchronous, but in some databases the commit can be deferred until the transaction has been replicated to the slave.



Distributed database patterns

1.2 Shared Nothing and Shared Disk

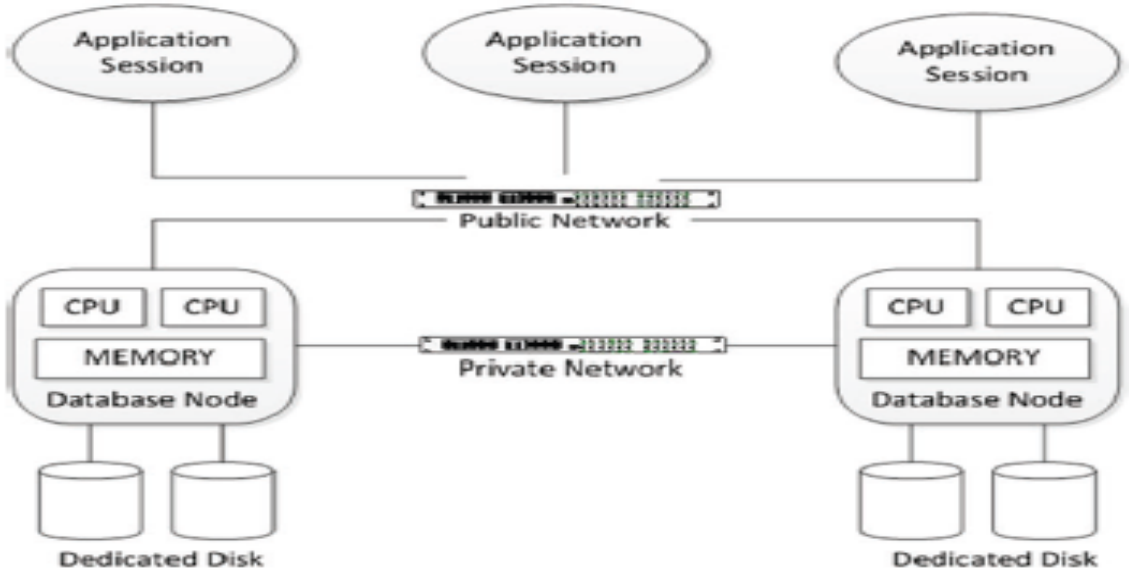
- The replication pattern for distributing database workloads works well to distribute read activity across multiple servers, but it does not distribute transactional write loads.
- Parallelizing a query across multiple database servers requires a new approach. Data warehousing vendors provided a solution to this problem by implementing a shared-nothing clustered database architecture
- A database server may be classified as:

- Shared-everything: In this case, every database process shares the same memory, CPU, and disk resources. Sharing memory implies that every process is on the same server and hence this architecture is a single-node database architecture.

Distributed database patterns

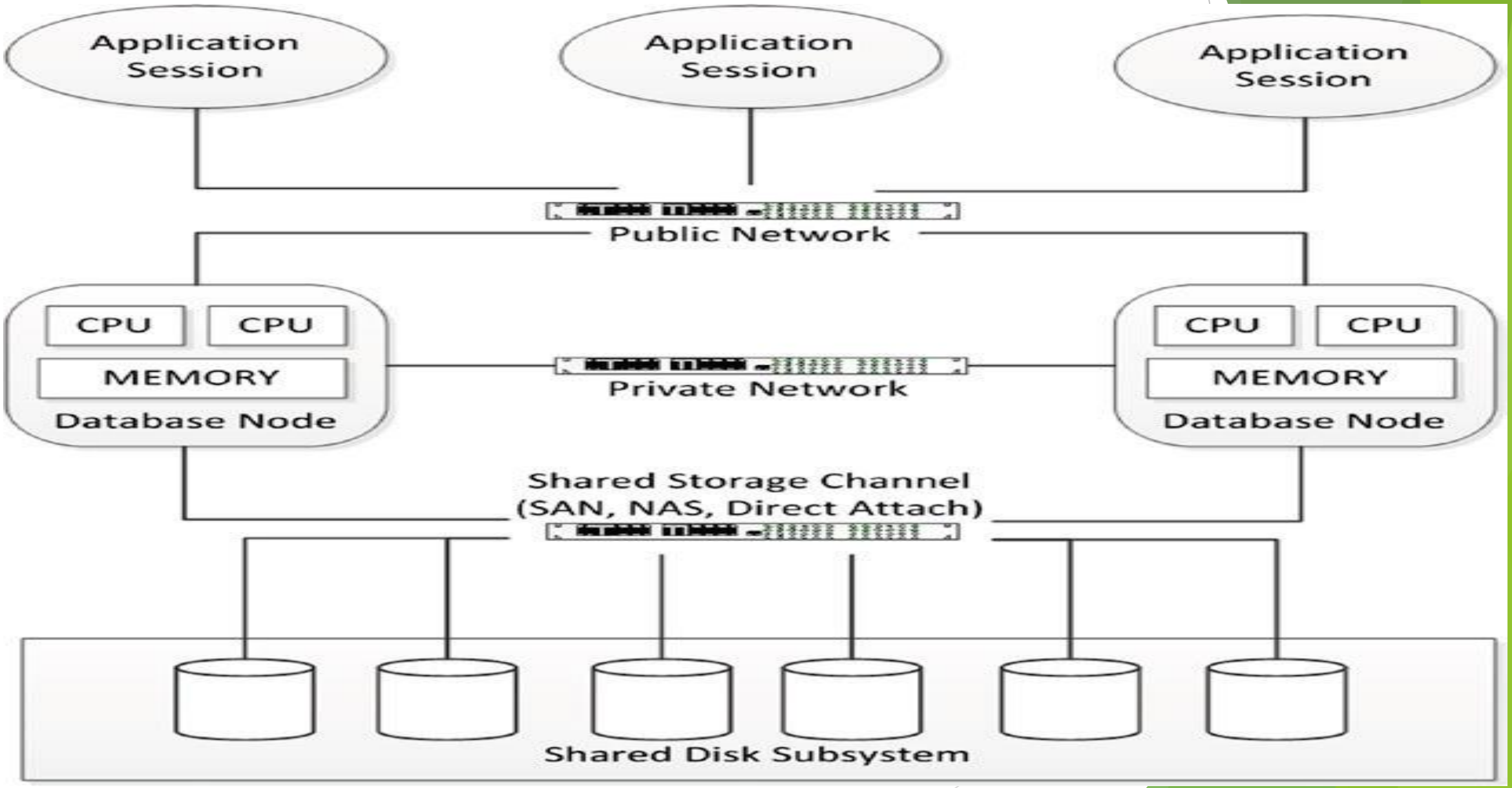
1.2 Shared Nothing and Shared Disk

- Shared-disk: In this case, database processes may exist on separate nodes in the cluster and have access to the CPU and memory of the server on which they reside.However, every process has equal access to disk devices, which are shared across all nodes of the cluster.
- Shared-nothing: In this case, each node in the cluster has access not only to its own memory and CPU but also to dedicated disk devices and its own subset of the database



shared-nothing database architecture

Distributed database patterns



Non relational distributed database

- In non relational database systems, ACID compliance is often not provided.
- There have been three broad categories of distributed database architecture adopted by next-generation databases. The three models are:
- Variations on traditional sharding architecture, in which data is segmented across nodes based on the value of a “shard key.”
- Variations on the Hadoop HDFS/HBase model, in which an “omniscient master” determines where data should be located in the cluster, based on load and other factors
- The Amazon Dynamo consistent hashing model, in which data is distributed across nodes of the cluster based on predictable mathematical hashing of a key value. We use MongoDB as an example of a sharding architecture, HBase as the example of an omniscient master, and Cassandra as an example of Dynamo-style consistent hashing.

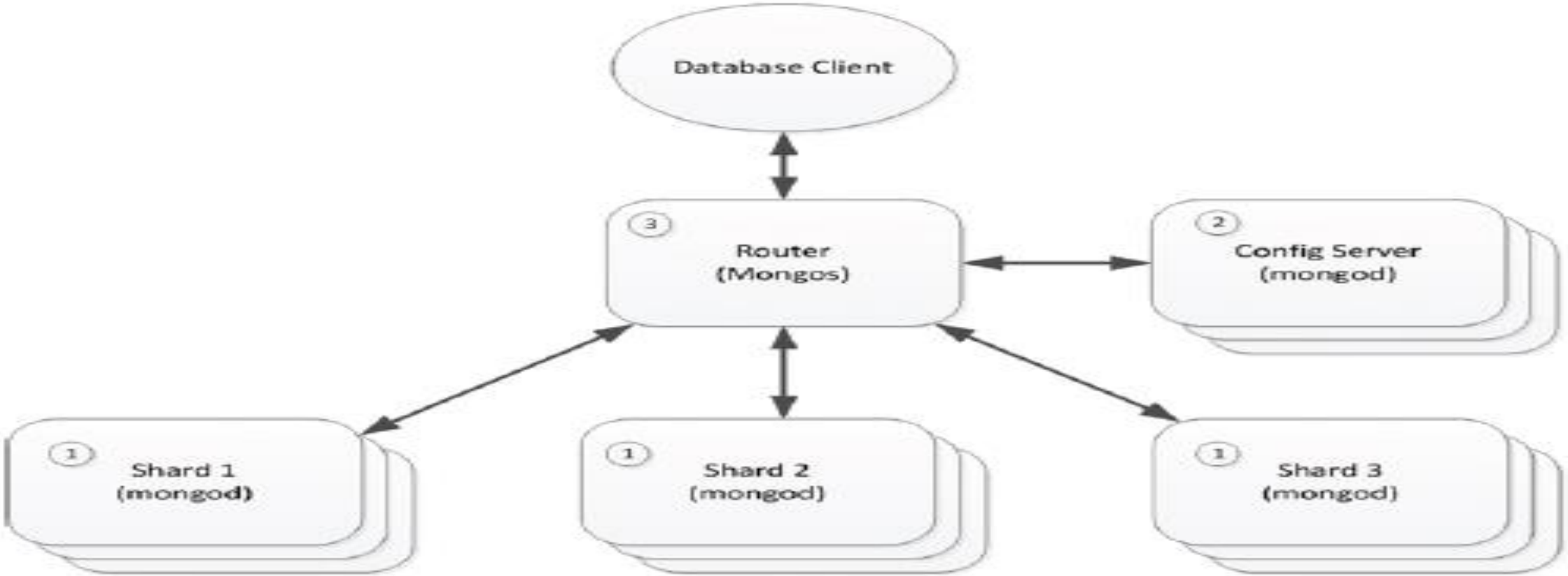
Introduction to MongoDB

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling. A record in MongoDB is a document, which is a data structure composed of field and value pairs.
- MongoDB consists of JSON,XML based documents
- MongoDB supports sharding to provide scale-out capabilities and replication for high availability. Sharding is a method for distributing data across multiple machines
- Sharding allows a logical database to be partitioned across multiple physical servers.
- In a sharded application, the largest tables are partitioned across multiple database servers.
- Each partition is referred to as a shard. This partitioning is based on a Key Value, such as a user ID.
- When operating on a particular record, the application must determine which shard will contain the data and then send the SQL to the appropriate server

Introduction to MongoDB

Introduction to MongoDB

3.1 Sharding



MongoDB sharding architecture

Introduction to MongoDB

3.1 Sharding

- Each shard is implemented by a distinct MongoDB database (1).
- A separate MongoDB database—the config server (2)—contains the metadata that can be used to determine how data is distributed across shards.
- A router process (3) is responsible for routing requests to the appropriate shard server.
- In MongoDB, a collection is used to store multiple JSON documents that usually have some common attributes. To shard a collection, we choose a ***shard key, which is one or more indexed attributes that will be used to determine the distribution of documents across shards***

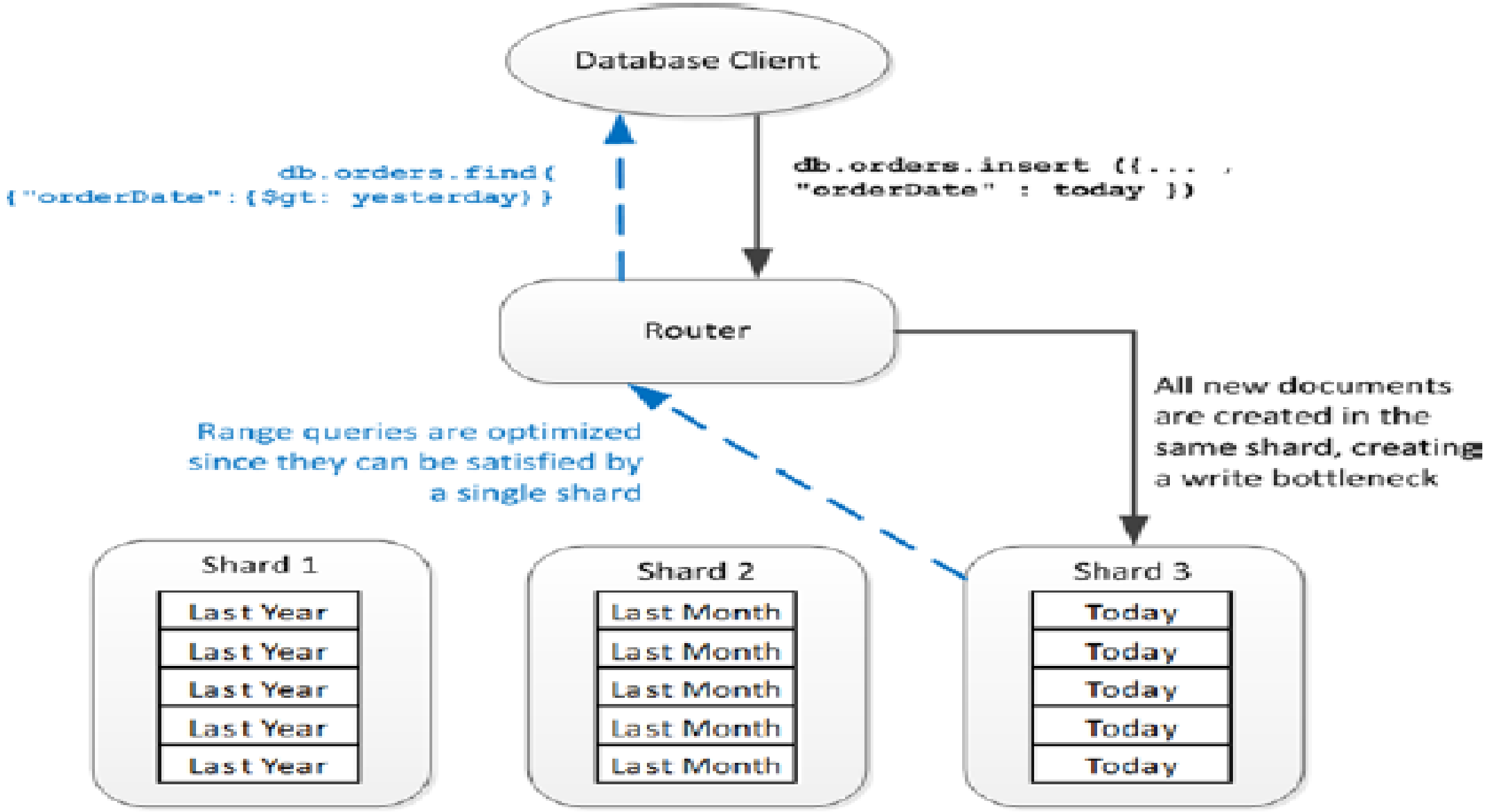
Introduction to MongoDB

3.1 Sharding

Sharding Mechanisms

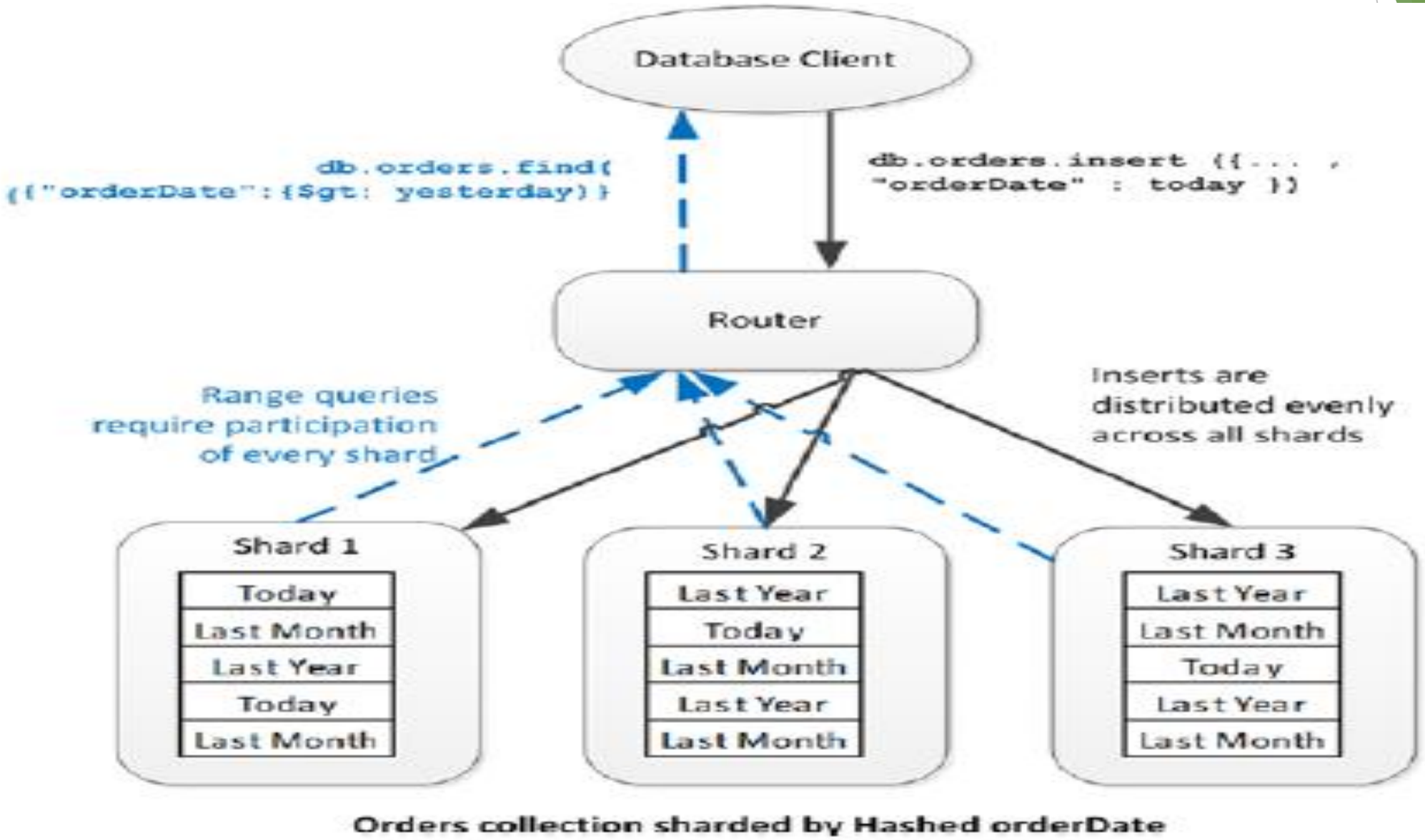
- Distribution of data across shards can be either range based or hash based.
- In range-based partitioning, each shard is allocated a specific range of shard key values.
- In hash-based sharding, the keys are distributed based on a hash function applied to the shard key.
- Range-based partitioning allows for more efficient execution of queries that process ranges of values, since these queries can often be resolved by accessing a single shard.
- Hash-based sharding requires that range queries be resolved by accessing all shards..

Introduction to MongoDB



Orders collection sharded by orderDate

Introduction to MongoDB



Orders collection sharded by Hashed orderDate

Introduction to MongoDB

- When range partitioning is enabled and the shard key is continuously incrementing, the load tends to aggregate against only one of the shards, thus unbalancing the cluster.
- With hash-based partitioning new documents are distributed evenly across all members of the cluster.
- Tag-aware sharding allows the MongoDB administrator to fine-tune the distribution of documents to shards by associating a shard with the tag, and associating a range of keys within a collection with the same tag

Introduction to MongoDB

Cluster Balancing

- MongoDB will periodically assess the balance of shards across the cluster and perform rebalance operations, if needed.
- The unit of rebalance is the shard chunk. Shards consist of chunks—typically 64MB in size—that contain contiguous values of shard keys (or of hashed shard keys).
- If a shard is added or removed from the cluster, or if the balancer determines that a shard has become unbalanced, it can move chunks from one shard to another.
- The chunks themselves will be split if they grow too large.

Introduction to MongoDB

Replication

- Sharding is almost always combined with replication so as to ensure both availability and scalability in a production MongoDB deployment.
- In MongoDB, data can be replicated across machines by the means of replica sets.
- A replica set consists of a primary node together with two or more secondary nodes. The primary node accepts all write requests, which are propagated asynchronously to the secondary nodes.
- The primary node is determined by an election involving all available nodes. To be eligible to become primary, a node must be able to contact more than half of the replica set
- The successful primary will be elected based on the number of nodes to which it is in contact, together with a priority value that may be assigned by the system administrator..

Introduction to MongoDB

Replication

- Setting a priority of 0 to an instance prevents it from ever being elected as primary. In the event of a tie, the server with the most recent optime—the timestamp of the last operation—will be selected
- The primary stores information about document changes in a collection within its local database, called the oplog.
- The primary will continuously attempt to apply these changes to secondary instances.
- Members within a replica set communicate frequently via heartbeat messages.
- If a primary finds it is unable to receive heartbeat messages from more than half of the secondaries, then it will renounce its primary status and a new election will be called.

Hbase

IV Hbase

4.1 Introduction

- HDFS implements a distributed file system using disks that are directly attached to the servers—Data Nodes—that constitute a Hadoop cluster. By default, data is replicated across three Data Nodes, one of which (if possible) is located on a separate server rack.

4.2 Tables, Regions, and RegionServers

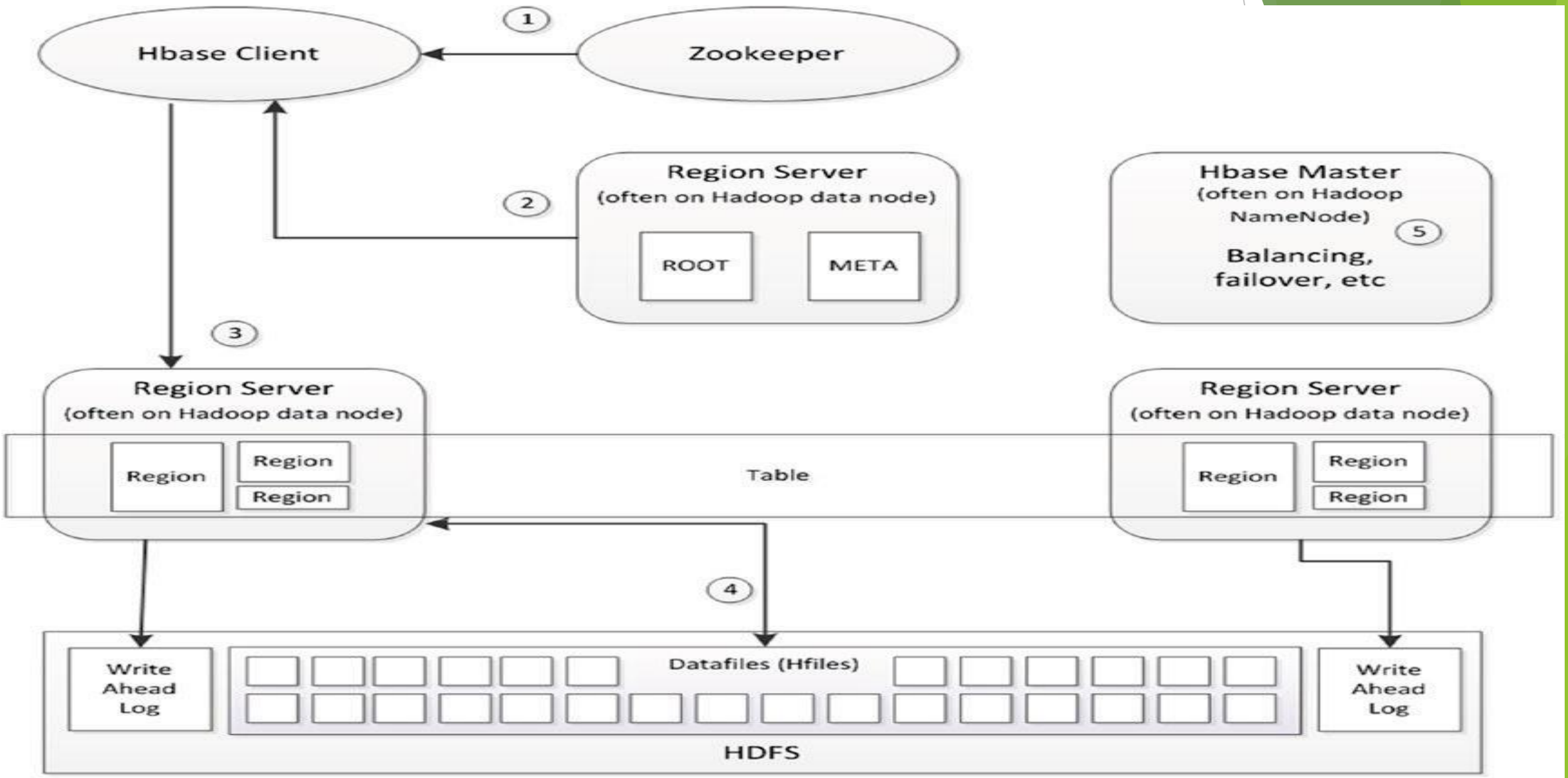
- Hbase tables are massive tabular datasets that are implemented on disk by a variable number of HDFS files called Hfiles.
- All rows in an Hbase table are identified by a unique row key. A table will be split into multiple horizontal partitions called regions. Each region consists of a contiguous, sorted range of key values
- Read or write access to a region is controlled by a Region Server

Hbase

4.2 Tables, Regions, and RegionServers

- Each Region Server normally runs on a dedicated host, and is typically co-located with the Hadoop Data Node. There will usually be more than one region in each Region Server.
- Each Hbase installation will include a Hadoop Zookeeper service that is implemented across multiple nodes. When an Hbase client wishes to read or write to a specific key value, it will ask Zookeeper for the address of the Region Server that controls the Hbase catalog.
- This catalog consists of the tables –ROOT- and .META., which identify the RegionServers that are responsible for specific key ranges. The client will then establish a connection with that Region Server and request to read or write the key value concerned.
- The Hbase master server performs a variety of housekeeping tasks. In particular, it controls the balancing of regions among RegionServers
- If a Region Server is added or removed, the master will organize for its regions to be relocated to other RegionServers

HBase


Hbase Client
1
Zookeeper
Region Server
(often on Hadoop data node)
2
ROOT
META
Hbase Master
(often on Hadoop
NameNode)
5
Balancing,
failover, etc
3
Region Server
(often on Hadoop data node)
Region
Region
Region
Table
Region Server
(often on Hadoop data node)
Region
Region
Region
4
Write
Ahead
Log
Datafiles (Hfiles)
Write
Ahead
Log
HDFS

Hbase

4.2 Tables, Regions, and RegionServers

- An Hbase client consults Zookeeper to determine the location of the Hbase catalog tables (1), which can be then be interrogated to determine the location of the appropriate RegionServer (2). The client will then request to read or modify a key value fromthe appropriate RegionServer (3). The RegionServer reads or writes to the appropriate disk files, which are located on HDFS (4).

4.3 Caching and Data Locality :

- The RegionServer includes a block cache that can satisfy many reads from memory, and a MemStore, which writes in memory before being flushed to disk. Each RegionServer has a dedicated write ahead log (WAL), which journals all writes to HDFS.

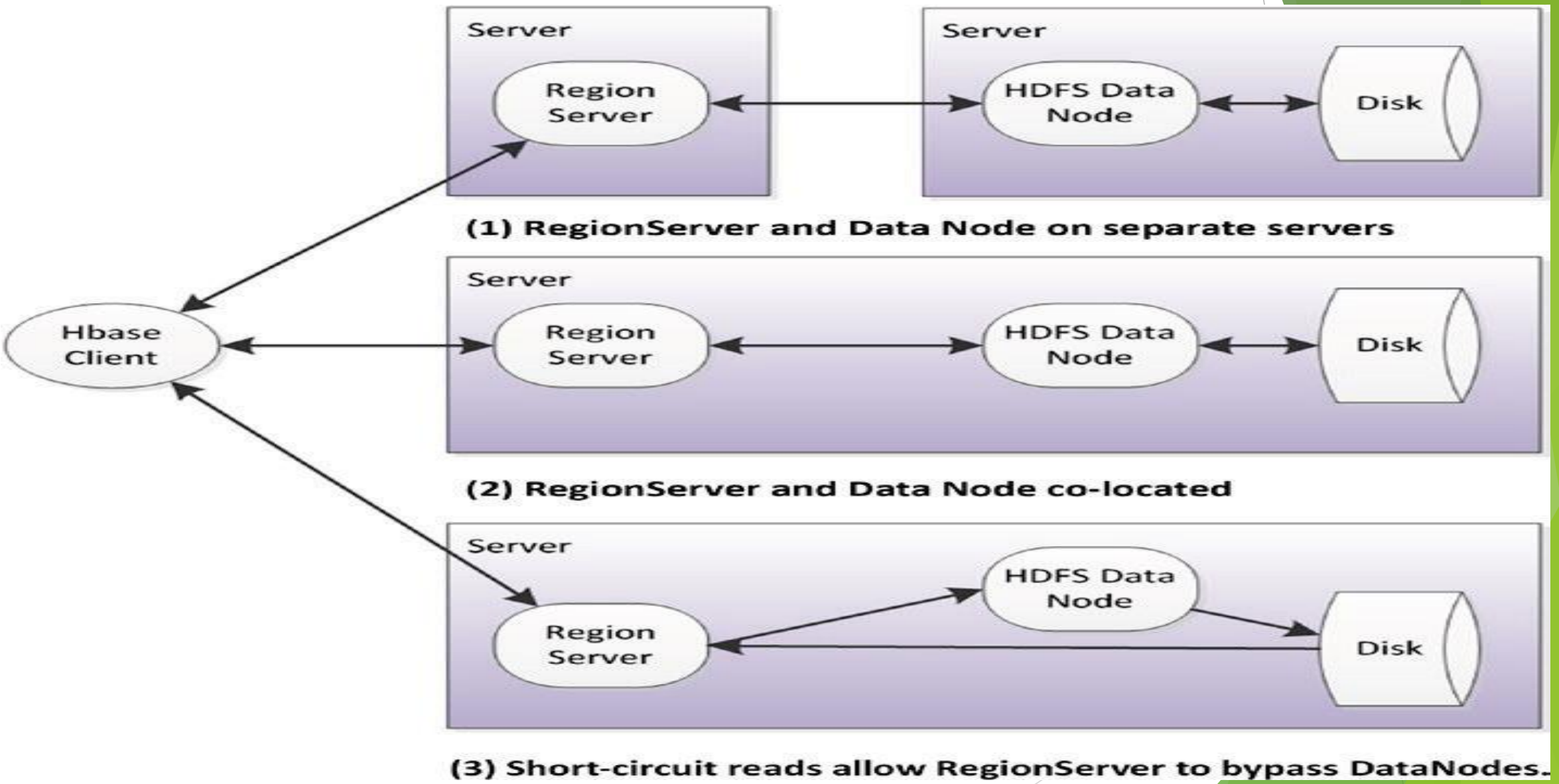
Hbase

4.3 Caching and Data Locality :

Data Locality :

- The three levels of data locality are shown in Figure. In the first configuration, the RegionServer and the Data Node are located on different servers and all reads and writes have to pass across the network.
- In the second configuration, the RegionServer and the Data Node are co-located and all reads and writes pass through the DataNode, but they are satisfied by the local disk. In the third scenario, short-circuit reads are configured and the RegionServer can read directly from the local disk.

Hbase


Server
Region Server
Server
HDFS Data Node
Disk
(1) RegionServer and Data Node on separate servers
Hbase Client
Server
Region Server
HDFS Data Node
Disk
(2) RegionServer and Data Node co-located
Server
HDFS Data Node
Region Server
Disk
(3) Short-circuit reads allow RegionServer to bypass DataNodes.

Hbase

4.4 Rowkey Ordering

- The Hbase region partitioning scheme requires that regions consist of contiguous ranges of row keys.
- When the row key contains some form of incrementing value , all write operations will be directed to a specific region and hence to a single RegionServer. This can create a bottleneck on write throughput.
- Hbase offers no internal mechanisms to mitigate this issue. It’s up to the application designer to construct a key that is either randomized—a hash of the timestamp, for instance—or is prefixed in some way with a more balanced attribute

Hbase

4.4 RegionServer Splits, Balancing, and Failure

- As regions grow, they will be split by the RegionServer as required.
- The new regions will remain controlled by the original RegionServer, but they are eligible for relocation during load-balancing operations.
- Hbase master node should balance regions across RegionServers.
- The master will periodically evaluate the balance of regions across all RegionServers, and should it detect an imbalance, it will migrate regions to another server.
- Rebalancing tends to result in a loss of data locality: when the RegionServer acquires responsibility for a new region, that region will probably be located on a remote data node.

Hbase

4.5 Region Replicas

- Region replicas allow for redundant copies of regions to be stored on multiple RegionServers. Should a RegionServer fail, these replicas can be used to service client requests.
- The original RegionServer serves as the master copy of the region. Read-only replicas of the region are distributed to other RegionServers

Cascandra

- Based on the Amazon Dynamo consistent hashing model, in which data is distributed across nodes of the cluster based on predictable mathematical hashing of a key value.

5.1 Gossip

- In Cassandra and other Dynamo databases, there are no specialized master nodes. Every node is equal and every node is capable of performing any of the activities required for cluster operation.
- In the absence of such a master node , Cassandra requires that all members of the cluster be kept up to date with the current state of cluster configuration and status. This is achieved by use of the gossip protocol
- Every second each member of the cluster will transmit information about its state and the state of any other nodes it is aware of to up to three other nodes in the cluster. In this way, cluster status is constantly being updated across all members of the cluster.

Cascandra

5.2 Consistent Hashing

- Cassandra and other dynamo-based databases distribute data throughout the cluster by using consistent hashing.
- The row key (analogous to a primary key in an RDBMS) is hashed. Each node is allocated a range of hash values, and the node that has the specific range for a hashed key value takes responsibility for the initial placement of that data.
- In the default Cassandra partitioning scheme, the hash values range from -263 to 263-1. Therefore, if there were four nodes in the cluster and we wanted to assign equal numbers of hashes to each node, then the hash ranges for each would be approximately as follows:

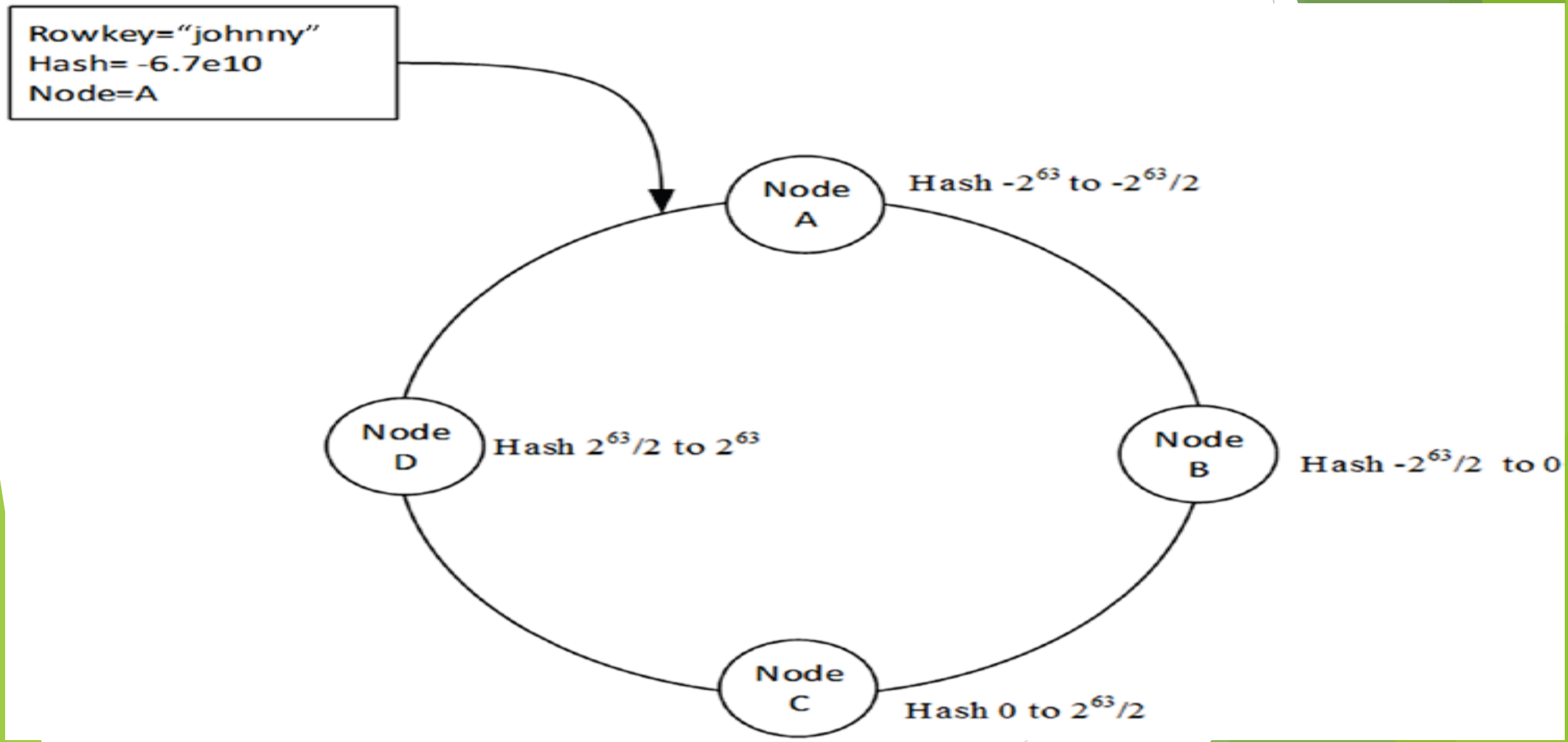
Node	Low Hash	High Hash
Node A	-2^{63}	$-2^{63}/2$
Node B	$-2^{63}/2$	0
Node C	0	$2^{63}/2$
Node D	$2^{63}/2$	2^{63}

Cascandra

5.2 Consistent Hashing

- We usually visualize the cluster as a ring: the circumference of the ring represents all the possible hash values, and the location of the node on the ring represents its area of responsibility.
- Figure given below illustrates simple consistent hashing: the value for a row key is hashed, which determines its position on “the ring.”Nodes in the cluster take responsibility for ranges of values within the ring, and therefore take ownership of specific row key values
- The four-node cluster in the Figure is well balanced because every node is responsible for hash ranges of similar magnitude.

Cascandra

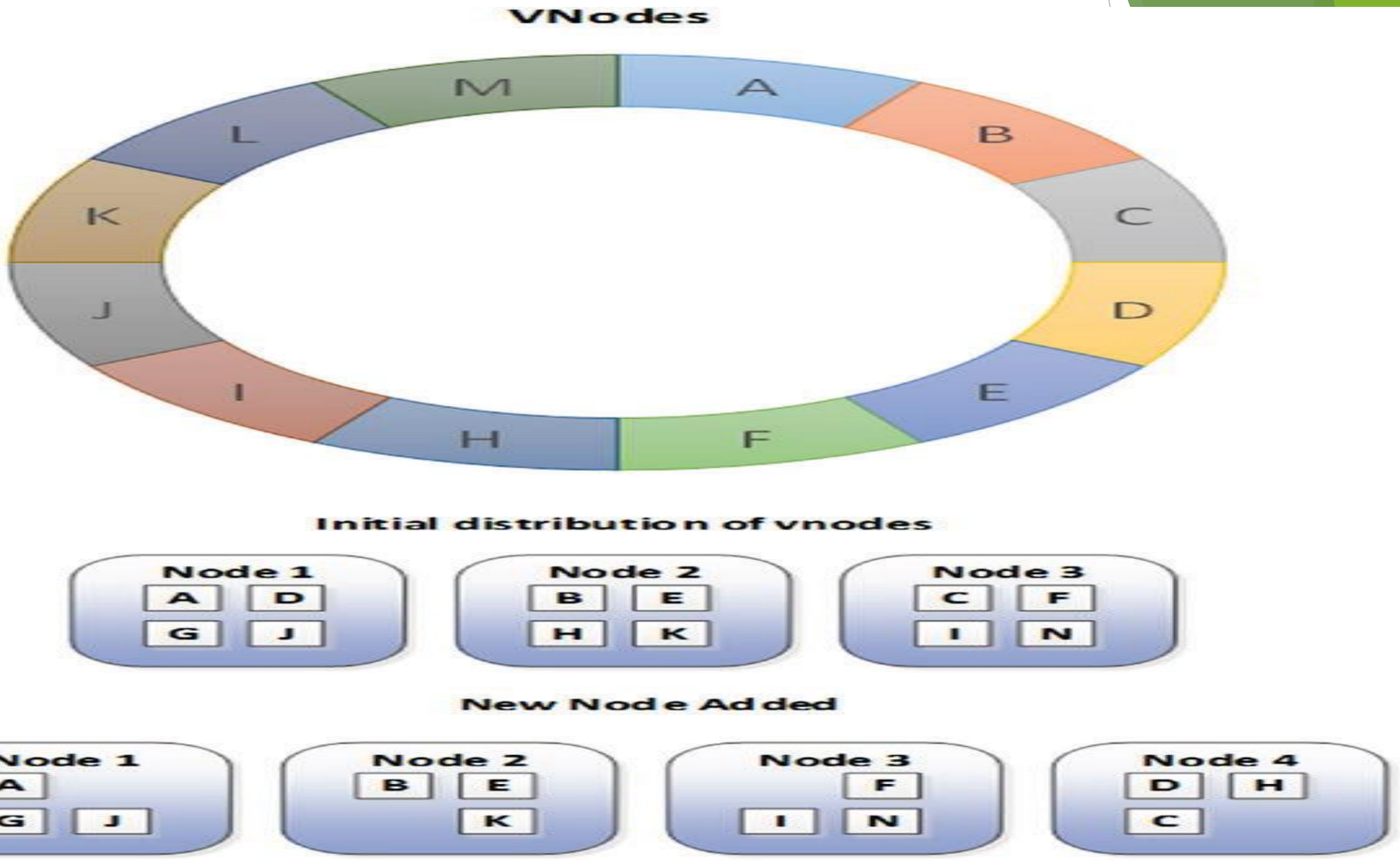

Rowkey="johnny"
Hash= -6.7e10
Node=A
Node A
Hash -2^63 to -2^63/2
Node B
Hash -2^63/2 to 0
Node C
Hash 0 to 2^63/2
Node D
Hash 2^63/2 to 2^63

Cascandra

Virtual Nodes:

- But when we add nodes there will be cluster unbalancing . Virtual nodes, implemented in Cassandra, provide a solution to this issue.
- When using virtual nodes, the hash ranges are calculated for a relatively large number of virtual nodes—256 virtual nodes per physical node, typically—and these virtual nodes are assigned to physical nodes.
- Now when a new node is added, specific virtual nodes can be reallocated to the new node, resulting in a balanced configuration with minimal overhead.

Cascandra


VNodes
M
A
L
B
K
C
J
D
I
E
H
F
Initial distribution of vnodes
Node 1
A
D
G
J
Node 2
B
E
H
K
Node 3
C
F
I
N
New Node Added
Node 1
A
G
J
Node 2
B
E
K
Node 3
F
I
N
Node 4
D
H
C

Cascandra

5.3 Order-Preserving Partitioning

- The default partitioner uses consistent hashing, as described in the previous section. Cassandra also supports order-preserving partitioners that distribute data across the nodes of the cluster as ranges of actual (e.g., not hashed) row keys.
- Requests for specific row ranges are isolated to specific machines, but it can lead to an unbalanced cluster and may create hotspots, especially if the key value is incrementing

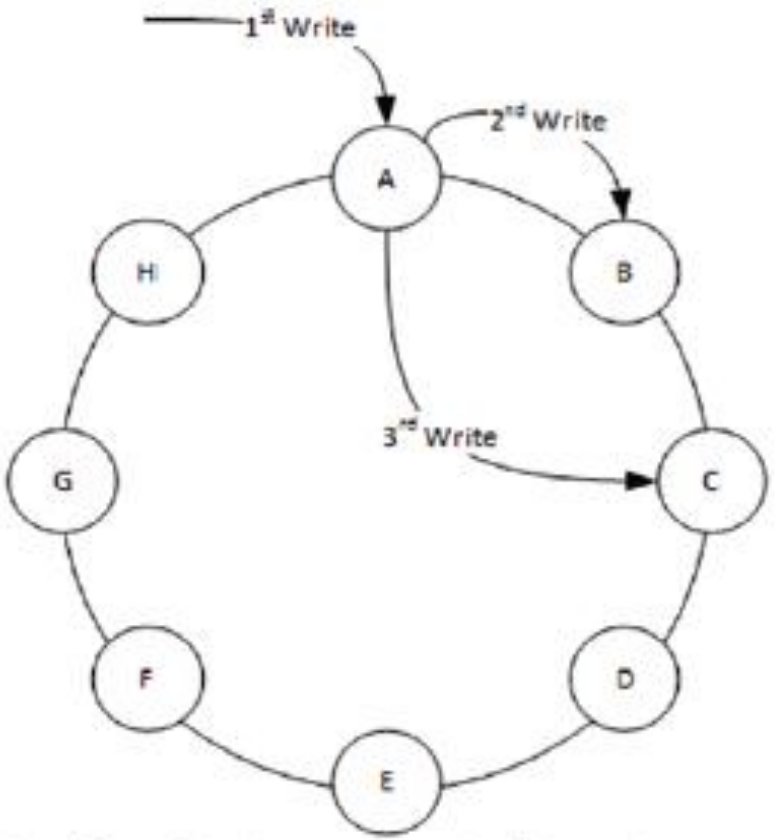
5.4 Replicas

- The consistent hashing algorithm determines where replicas of data items are stored. The node responsible for the hash range that equates to a specific row key value is called the coordinator node.
- The coordinator is responsible for ensuring that the required number of replica copies of the data are also written. The number of nodes to which the data must be written is known as the replication factor, and is the “N” in the NWR notation.

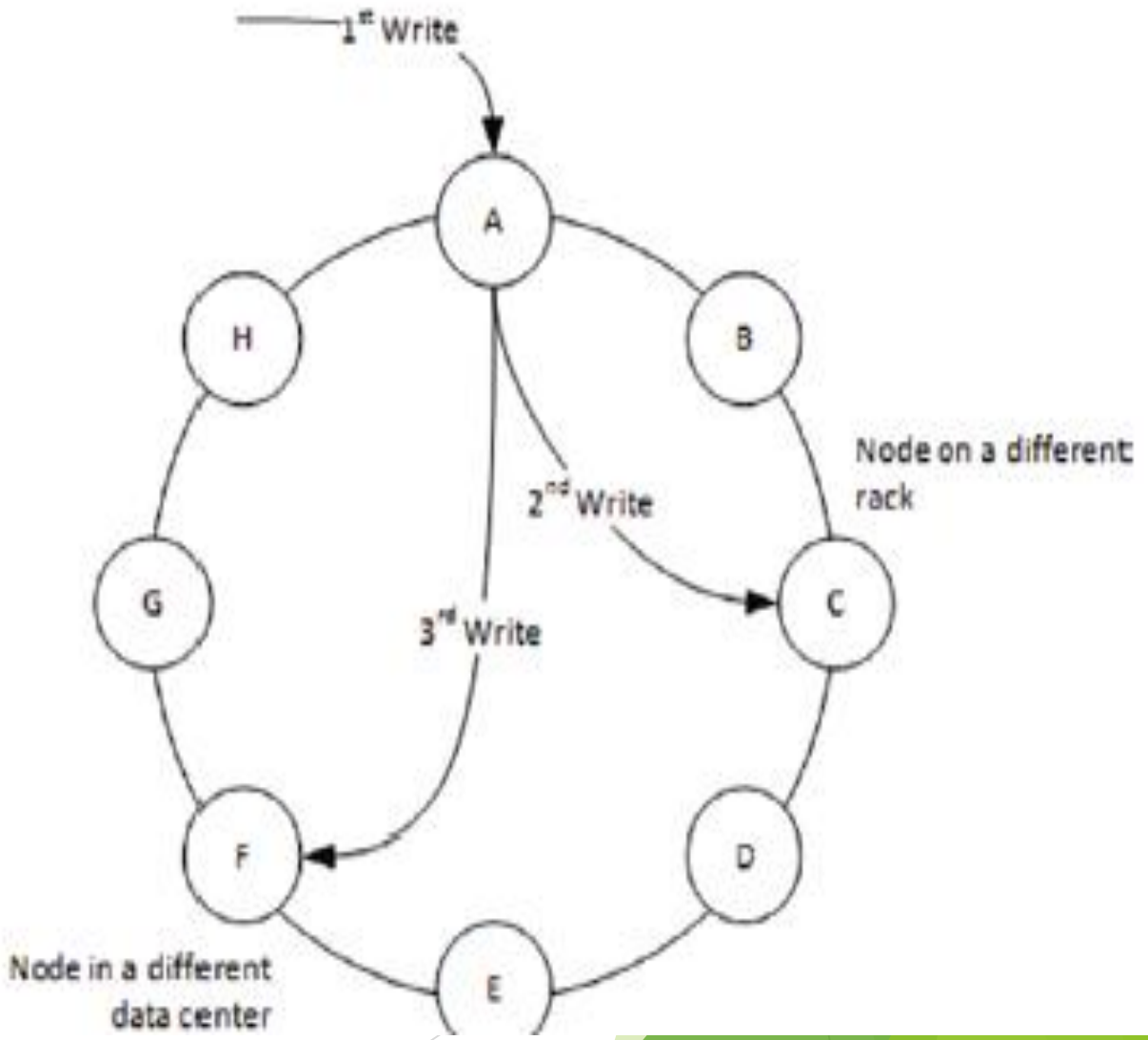
Cascandra

- By default, the coordinator will write copies of the data to the next N-1 nodes on the ring. So if the replication factor is 3, the coordinator will send replicas of the data item to the next two nodes on the ring.
- In this scenario, each node will be replicating data from the previous two nodes on the ring and replicating to the next two nodes on the ring.
- This simple scheme is referred to as the simple replication strategy.
- Cassandra also allows you to configure a more complex and highly available scheme.
- The Network Topology Aware replication strategy ensures that copies will be written to nodes on other server racks within the same data center, or optionally to nodes in another data center altogether

Cascandra



Simple replication strategy: Replicas written to next adjacent nodes on the ring



Cascandra

5.5 Snitches

- Cassandra uses snitches to help optimize read and write operations. A variety of snitches may be configured.
- The simple Snitch returns only a list of nodes on the ring. This is sufficient for the simple replication strategy we discussed in the previous section.
- The Rack Inferring Snitch uses the IP addresses of hosts to infer their rack and data center location. This snitch supports the network aware replication strategy.
- The Property FileSnitch uses data in the configuration file rather than IP addresses to determine the data center and rack topology.

CAP Theorem

- The CAP theorem says that in a distributed database system, we can have at most only two of Consistency, Availability, and Partition tolerance
- Consistency means that every user of the database has an identical view of the data at any given instant
- Availability means that in the event of a failure, the database remains operational
- Partition tolerance means that the database can maintain operations in the event of the network's failing between two segments of the distributed system

CAP Theorem

- The CAP theorem, also known as Brewer's theorem, states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:
- Consistency (all nodes see the same data at the same time)
- Availability (a guarantee that every request receives a response about whether it was successful or failed)
- Partition tolerance (the system continues to operate despite arbitrary message loss or failure of part of the system)
- According to the theorem, a distributed system cannot satisfy all three of these guarantees at the same time.

Cassandra and CAP

- Cassandra is typically classified as an AP system, meaning that availability and partition tolerance are generally considered to be more important than consistency in Cassandra

CAP Theorem