Distributed Database

CS 4750
Database Systems

[Silberschatz, Korth, Sudarshan, "Database System Concepts," 7th Edition, Ch. 21]
[Pattamsetti, "Distributed Computing in Java 9," Ch. 6]
[Ricardo and Urban, "Databases illuminated," 3rd ed., Ch. 10]

Distributed Database (DDB)

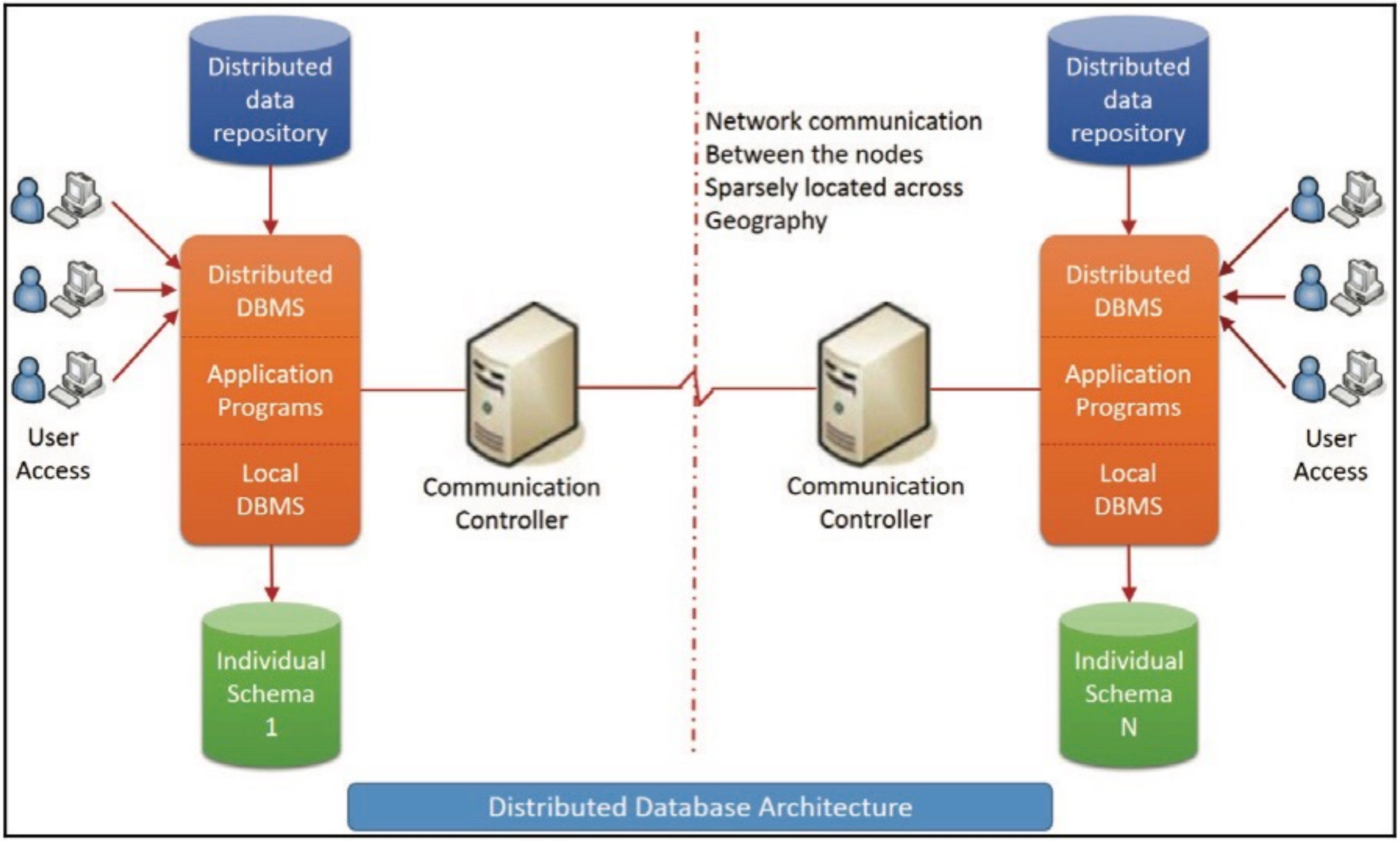
- A collection of multiple, logically interconnected databases that are physically distributed over a computer network on different sites
- Data are physically stored across multiple sites, managed by a DBMS that is independent of the other site
- Data at any site available to users at other sites
- Sites may be far apart, linked by some forms of telecommunication lines (secure lines or Internet)
- Sites that are close together may be linked by a local area network (LAN)

Distributed databases – focus on database storage and location transparency

Distributed Database Management System (DDBMS)

- A centralized software system that manages the DDB
- Synchronizes the databases periodically
- Provides an access mechanism that makes the distribution transparent to the users (as if it were all stored in a single location)
- Ensures that the data modified at any remote site is universally updated
- Supports a huge number of users simultaneously
- Maintains data integrity of the databases

Distributed Database Architecture



[Ref: Pattamsetti, "Distributed Computing in Java 9," p. 168]

Challenges

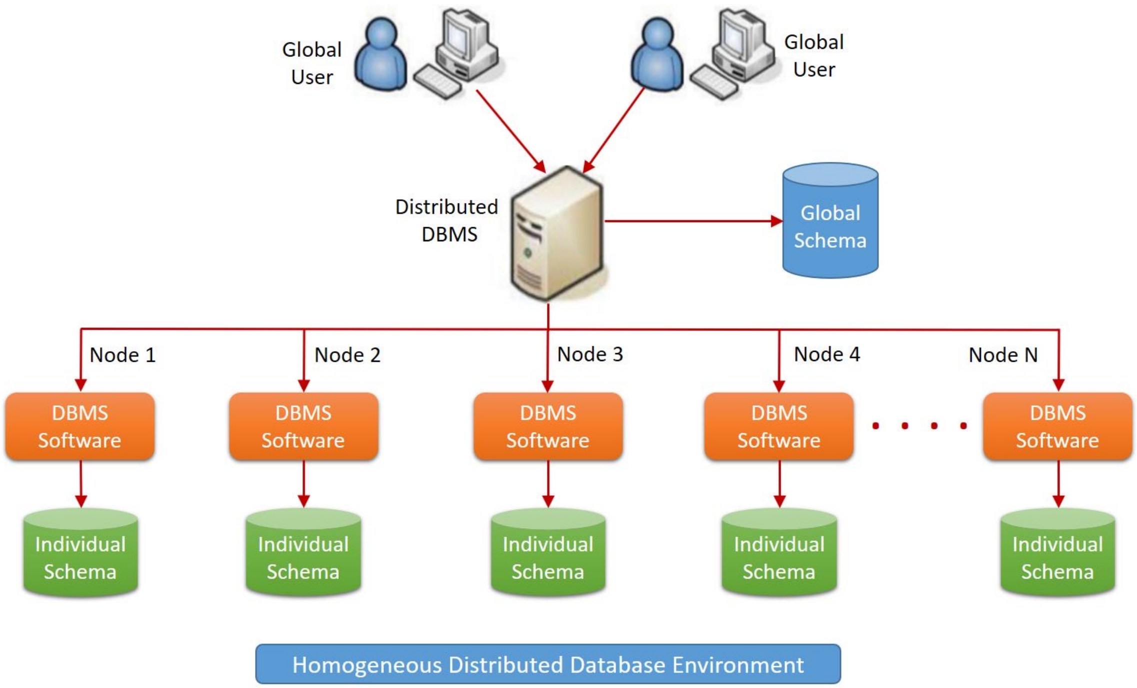
- Security: due to the Internet usage
- Consistency issues: databases must be synchronized periodically to ensure data integrity
- Increased storage requirements: due to replication of databases
- Multiple location access: transactions may access data at one or more sites

Distributed Strategies

Based on the organizational needs and information split and exchange requirements, the distributed database environment can be designed in two ways:

- **Homogeneous**
  - Use the same DBMS for all database nodes that take part in the distribution
- **Heterogeneous**
  - May use a diverse DBMS for some of the nodes that take part in the distribution

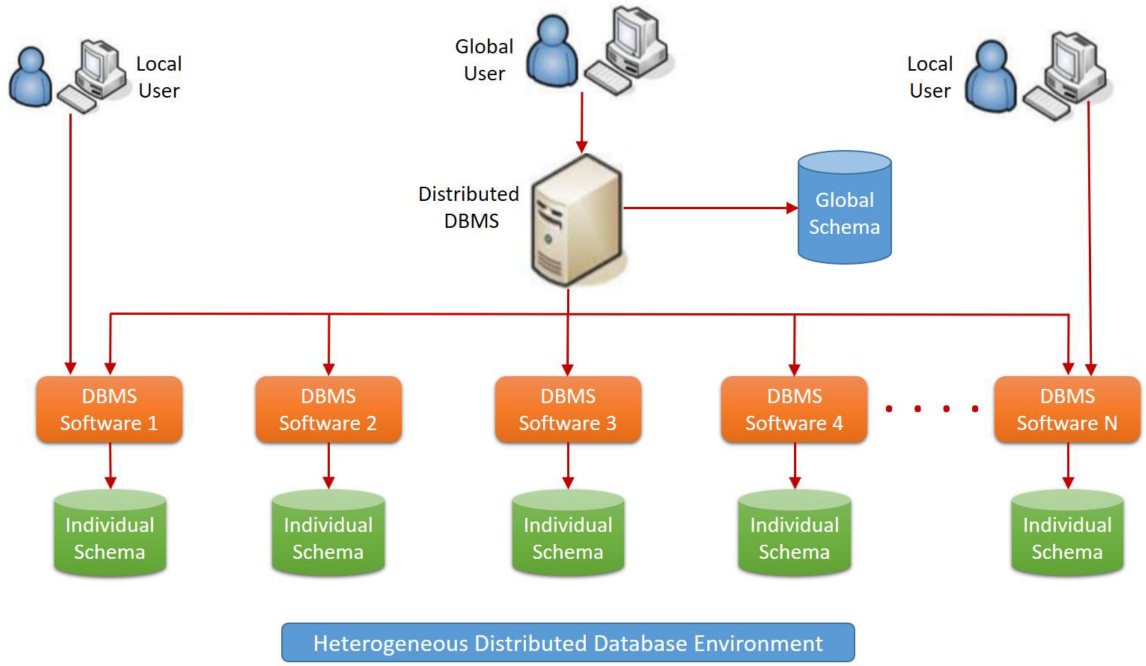
Homogeneous Distributed DB



- Information is distributed between all the nodes
- The same DBMS and schema are used across all the databases
- The distributed DBMS controls all information
- Every global user must access the information from the same global schema controlled by the distributed DBMS
- A combination of all the individual DB schemas makes the global schema

[Ref: Pattamsetti, "Distributed Computing in Java 9," p. 161]

Heterogeneous Distributed DB



- Information is distributed between all the nodes
- Different DBMS and schemas may be used across the databases
- Local users (interacting with one of the individual database) can access the corresponding DBMS and schema
- Users who want to access the global information can communicate with the distributed DBMS, which has a global schema (a combination of all the individual DB schemas)

[Ref: Pattamsetti, "Distributed Computing in Java 9," p. 163]

Distributed DB Setup Method

- The process of setting up the distributed DB environment involves a thorough analysis and design

- Ongoing and future information maintenance must be determined
  - Synchronous: information across all nodes should be kept in sync all the time
  - Asynchronous: information is replicated at multiple nodes to make it available for other nodes

- Once the analysis for a specific distributed DB environment is made, the setup can be performed in one of the following ways:
  - Replication
  - Fragmentation/partitioning (horizontal or vertical)
  - Hybrid setup

Replication

- Maintain multiple copies of the database instances, stored in different sites
- Easy and minimum risk process as the information is copied from one instance to another without a logical separation
- Each individual node has the complete information
- Efficient in accessing the information without having network traversals and reduces the risk of network security
- Fast retrieval
- Increase fault tolerance
- Require more storage space
- Take longer to synchronize all the nodes when the information across all the nodes needs to be updated

Fragmentation (or Partition)

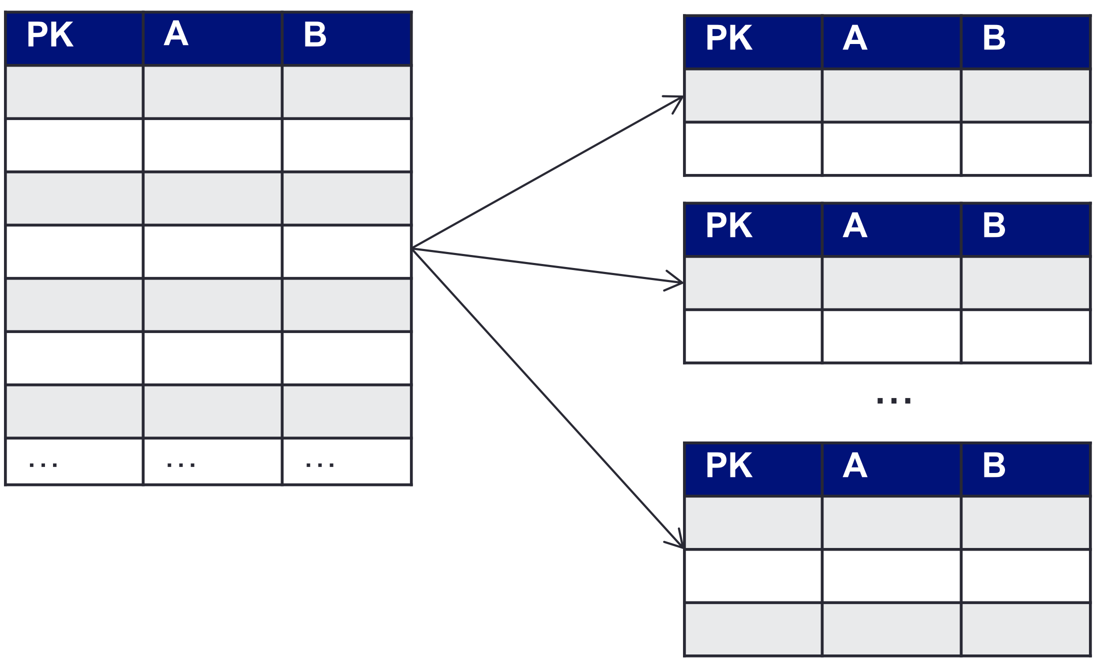
- One copy of each data item, distributed across nodes
- Split a database into disjoint fragments (or parts)
- Fragments can be
  - Vertical table subsets (formed by RA projection)
  - Horizontal subsets (formed by RA selection)

Horizontal Fragmentation

- Splitting the **rows of a table** (or a relation between two or more nodes, containing databases) to form a distributed database – "**split by region**"

- Each individual database has a set of rows that belong to the table or relation that belongs to the specific database

N nodes



R_1, $-\text{inf} < PK <= v_1$

R_2, $v_1 < PK <= v_2$

R_N, $v_N < PK < \text{inf}$

Example: Horizontal Fragmentation

stuId	lastName	firstName	major	credits
S1001	Smith	Tom	History	90
S1002	Chin	Ann	Math	36
S1005	Lee	Perry	History	3
S1010	Burns	Edward	Art	63
S1013	McCarthy	Owen	Math	0
S1015	Jones	Mary	Math	42
S1020	Rivera	Jane	CSC	15
…	…	…	…	…

$\sigma_{major=\text{“Math”}}$ (students)

stuId	lastName	firstName	major	credits
S1002	Chin	Ann	Math	36
S1013	McCarthy	Owen	Math	0
S1015	Jones	Mary	Math	42
…	…	…	…	…

$\sigma_{major=\text{“History”}}$ (students)

stuId	lastName	firstName	major	credits
S1001	Smith	Tom	History	90
S1005	Lee	Perry	History	3
…	…	…	…	…

$\sigma_{major=\text{“Art”}}$ (students)

stuId	lastName	firstName	major	credits
S1010	Burns	Edward	Art	63
…	…	…	…	…

$\sigma_{major=\text{“CSC”}}$ (students)

stuId	lastName	firstName	major	credits
S1020	Rivera	Jane	CSC	15
…	…	…	…	…

…

[Example adapted from Ricardo and Urban, “Databases Illuminated,” fragmentation example, p. 452] – this example has been simplified

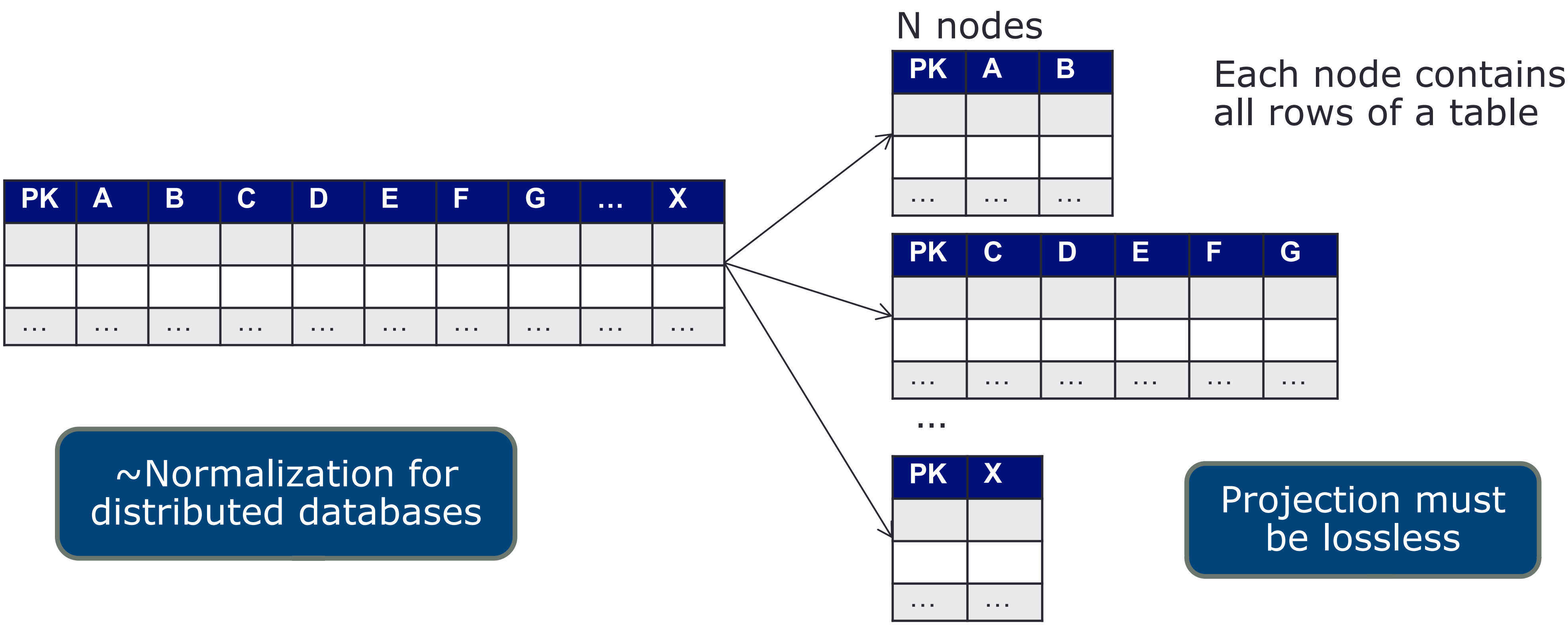
Horizontal Fragmentation

- The information access is efficient
- Best if partitions are uniform
- Optimal performance as the local data are only stored in a specific database
- Allow parallel processing on fragments
- More secure as the information belonging to the other location is not stored in the database
- If a user wants to access some of the other nodes or a combination of node information, the access latency varies.
- If there is a problem with a node or a network, the information related to that node becomes inaccessible to the users

Vertical Fragmentation

- (aka normalization process in distributed database setup)

- Splitting the **columns** of a table (or a relation between two or more nodes, containing databases) to form a distributed database while keeping a copy of the base column (primary key) to uniquely identifying each record – **“split by purpose”**



Example: Vertical Fragmentation

stuId	lastName	firstName	major	credits	…
S1001	Smith	Tom	History	90	…
S1002	Chin	Ann	Math	36	…
S1005	Lee	Perry	History	3	…
S1010	Burns	Edward	Art	63	…
S1013	McCarthy	Owen	Math	0	…
S1015	Jones	Mary	Math	42	…
S1020	Rivera	Jane	CSC	15	…
…	…	…	…	…	…

Notice stuId in all fragments

$\pi_{\text{stuId, lastName, firstName}}$(students)

stuId	lastName	firstName
S1001	Smith	Tom
S1002	Chin	Ann
S1005	Lee	Perry
S1010	Burns	Edward
S1013	McCarthy	Owen
S1015	Jones	Mary
S1020	Rivera	Jane
…	…	…

$\pi_{\text{stuId, major}}$(students)

stuId	major
S1001	History
S1002	Math
S1005	History
S1010	Art
S1013	Math
S1015	Math
S1020	CSC
…	…

$\pi_{\text{stuId, credits}}$ (students)

stuId	credits
S1001	90
S1002	36
S1005	3
S1010	63
S1013	0
S1015	42
S1020	15
…	…

…

[Example adapted from Ricardo and Urban, "Databases Illuminated," fragmentation example, p. 452] – this example has been simplified

Vertical Fragmentation

- Appropriate if each of the organizational units located in different geographies have separate operations
- Partition based on behavior and function that each node performs
- Best if partitions are uniform
- Part of the tuple is stored where it is most frequently accessed
- Allow parallel processing on fragments
- Poorly chosen columns to split can lead to node bottleneck
- The aggregation of the data involves complex queries with joins across the location database, as no replication is made for non-primary keys

Correctness of Fragmentation

- **Completeness**
  - Decomposition of a relation R into R_1, R_2, …, R_n is complete if and only if each data item in R can also be found in some R_i

- **Reconstruction**
  - If a relation R is decomposed into R_1, R_2, …, R_n, reconstructing R_1, R_2, …, R_n should result in the original R

- **Disjointness**
  - If a relation R is decomposed into R_1, R_2, …, R_n and data item d is in R_i, then d should not be in any other fragment R_j where $i <> j$

Hybrid Setup

- Involve a combination of replication and fragmentation
- Relation is partitioned into several fragments
- Some information is replicated across the database nodes
- Data administrators play a crucial role to choose the right combination to ensure data integrity and security

revisit

RDBMS and ACID Properties

Four properties of transactions that a DBMS follows to handle concurrent access while maintaining consistency

A C I D

Atomicity
- All or nothing

Consistency
- Start with consistent state, ends with consistent state

Isolation
- Concurrent transactions are isolated, executed without interference

Durability
- Committed transaction is persistent – recoverable if the system fails

> ACID work in a centralized database system, not in a distributed database system

Threats on ACID Properties

- While distributed database system has many advantages, it imposes a threat on ACID properties

- Consistency in database (ACID)
  - Database relies on a set of integrity constraints
  - DBMS executes each transaction to ensure Atomicity and Isolation and thus maintaining a consistent state

- Consistency in distributed database system with replication
  - Strong consistency:

Final state from a schedule with read and write operations on a replicated object	=	Final state from a schedule on a single copy of the object with order of operations from a single site preserved

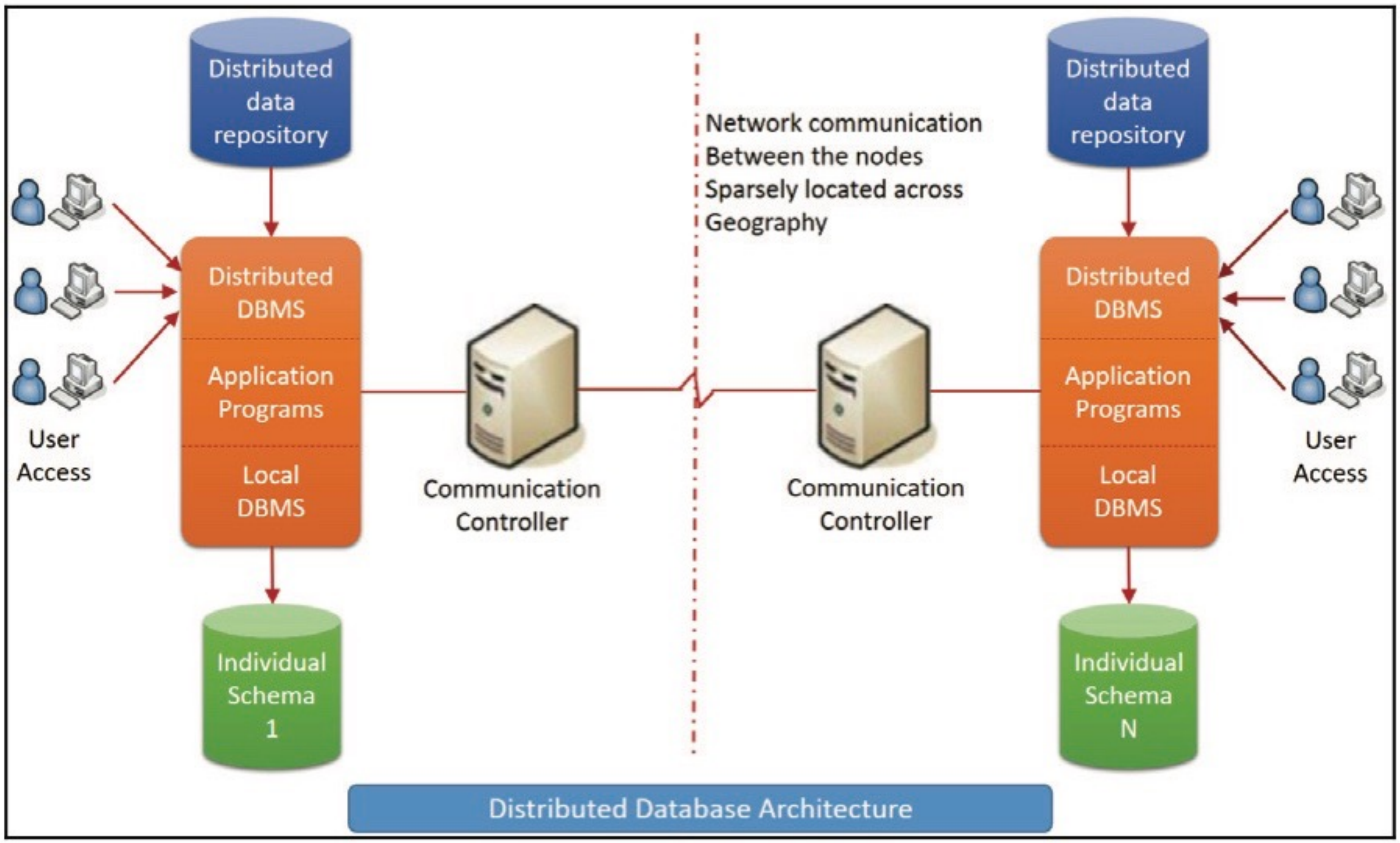
  - Weak consistency: (several forms)

CAP Theorem

- **Consistency** -- All copies (across nodes) have the same value
- **Availability** -- System can still function even if some nodes fail
- **Partition tolerance** -- System can function even if communication between nodes (the partitions reside) fails
  - Network can break into two or more parts, each with active systems that communicate with the other parts
- Must have **exactly two** of the three properties for any system
- Very large system will partition by default, thus choose one of consistency or availability
  - Traditional database – choose consistency
  - Most web apps – choose availability (except some specific/important parts such as order/payment processing)

CAP: Example Combination

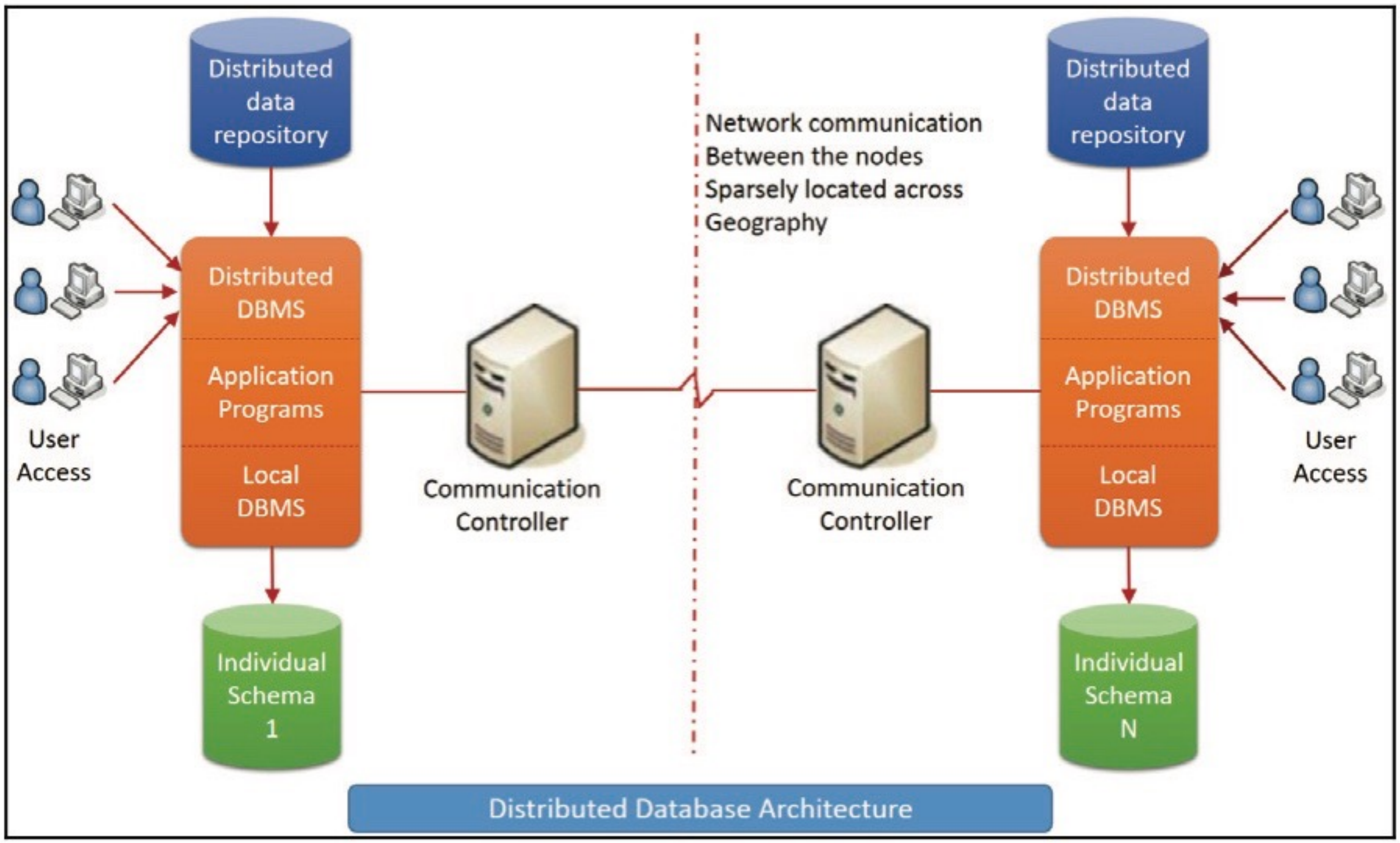
- **Consistency/Partition tolerance**
  - Queries are executed on one site. Then they are passed to all other sites, which then execute the queries.



[Ref: Pattamsetti, "Distributed Computing in Java 9," p. 168]

CAP: Example Combination

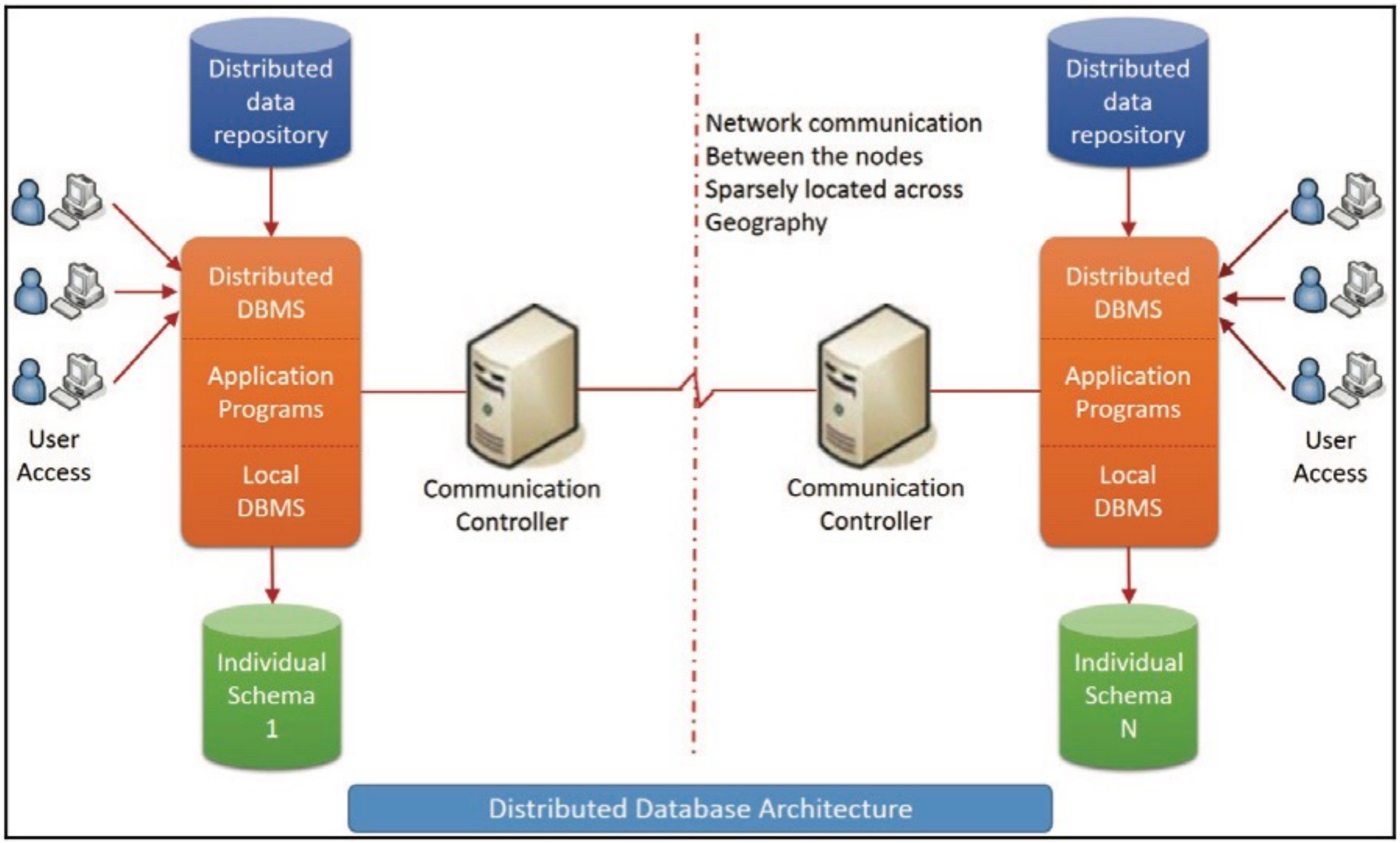
- **Availability/Partition tolerance**
  - Each site provides services independently. No impact if a network goes down or other sites fail. – resulting in inconsistent DBs



[Ref: Pattamsetti, "Distributed Computing in Java 9," p. 168]

CAP: Example Combination

- **Consistency/Availability**
  - Each site provides services independently as its own system.



[Ref: Pattamsetti, "Distributed Computing in Java 9," p. 168]

Threats on CAP

- Only two of the three properties are guarantees:
  - Consistency – every read receives the most recent write or an error
  - Availability – every request must respond with a non-error
  - Partition tolerance – continued operation in presence of dropped or delayed message

- Distributed RDBMS – partition tolerance + consistency

Intended to be highly consistent – but may sacrifice some consistency to boost availability

- NoSQL systems – partition tolerance + availability

Intended to be highly available – but may sacrifice some availability to boost consistency

With the growth of data → achieving CAP is very difficult. Instead of using ACID or CAP, use BASE (a more relaxed set of properties)

BASE Consistency Model

- With the enormous growth in data, achieving ACID or CAP becomes very difficult.
- A more relaxed set of properties is BASE
- **Basically Available, Soft state, Eventually consistent**

Most failures do not cause a complete system outage

System is not always write-consistent

Data will eventually converge to agreed values

- Key idea:
  - Databases may not all be in the same state at the same time ("soft state")
  - After synchronization is complete, the state will be consistent

Wrap-Up

- Distributed Database Systems → database scaling

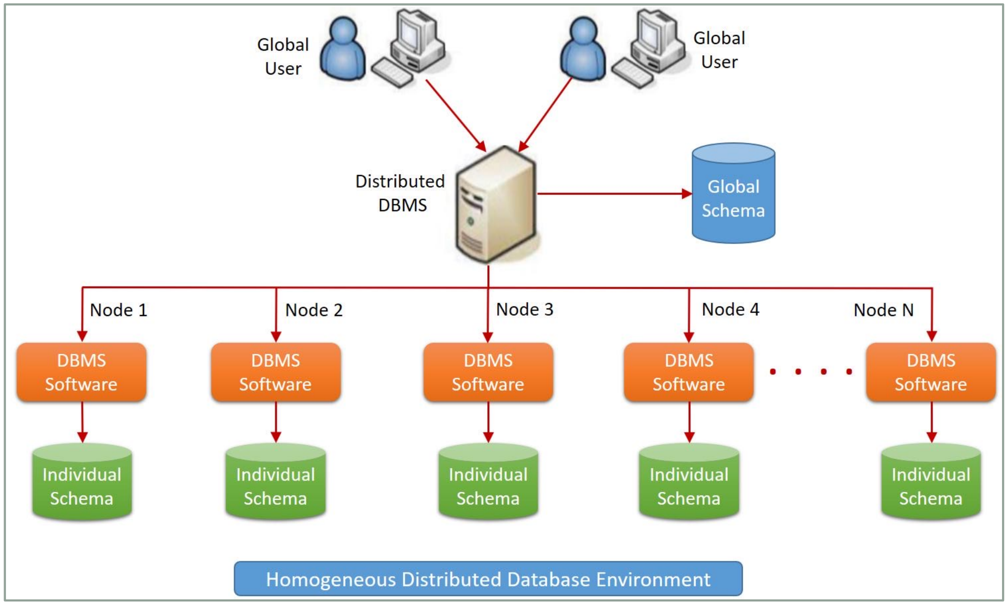
Replication

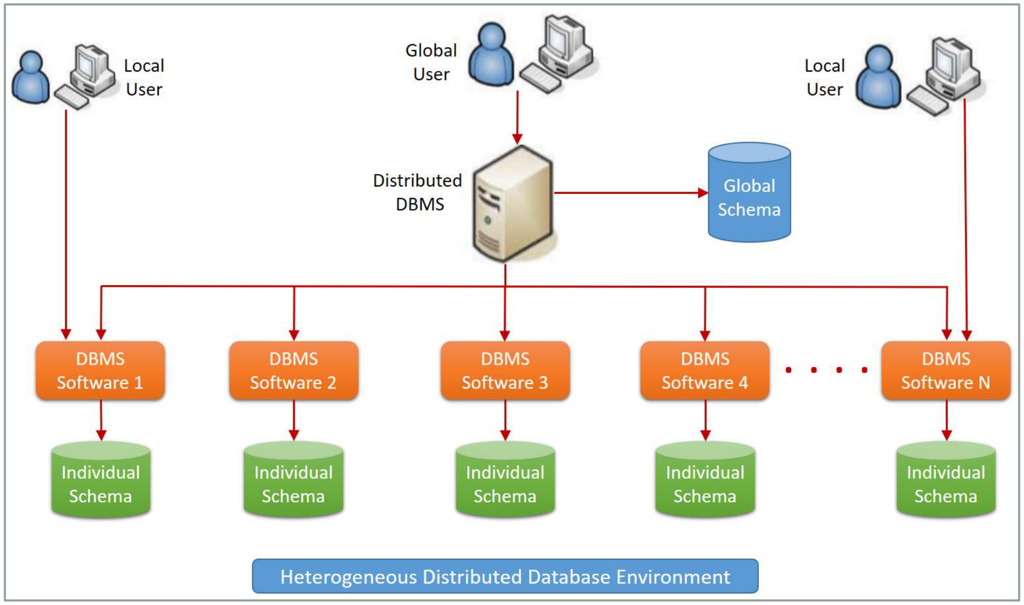
- Multiple copies of each database partition
- Improves fault tolerance
- Read performance ok
- Write performance suffers

Fragmentation

- Multiple machines to distribute data
- Write performance ok
- Read performance suffers

Wrap-Up (2)





- Fragmentation: need to coordinate operations across fragments
- Replication: need to synch to prevent inconsistent version
- Achieving ACID is challenging → use CAP in distributed DB

ACID work in a centralized database system,
not in a distributed database system

[Ref: images from Pattamsetti, "Distributed Computing in Java 9"]

Wrap-Up (3)

- RDBMS – intended to be highly consistent (boost availability by sacrificing some consistency)

- NoSQL – intended to be highly available (boost consistency by sacrificing some availability)

- Relational database systems – ACID
- Distributed database systems – CAP
- NoSQL systems – BASE

- Most applications compromise, depending business logic
  - Consistency / availability
  - Scalability
  - Usability
  - Analysis requirements

No silver-bullet !!