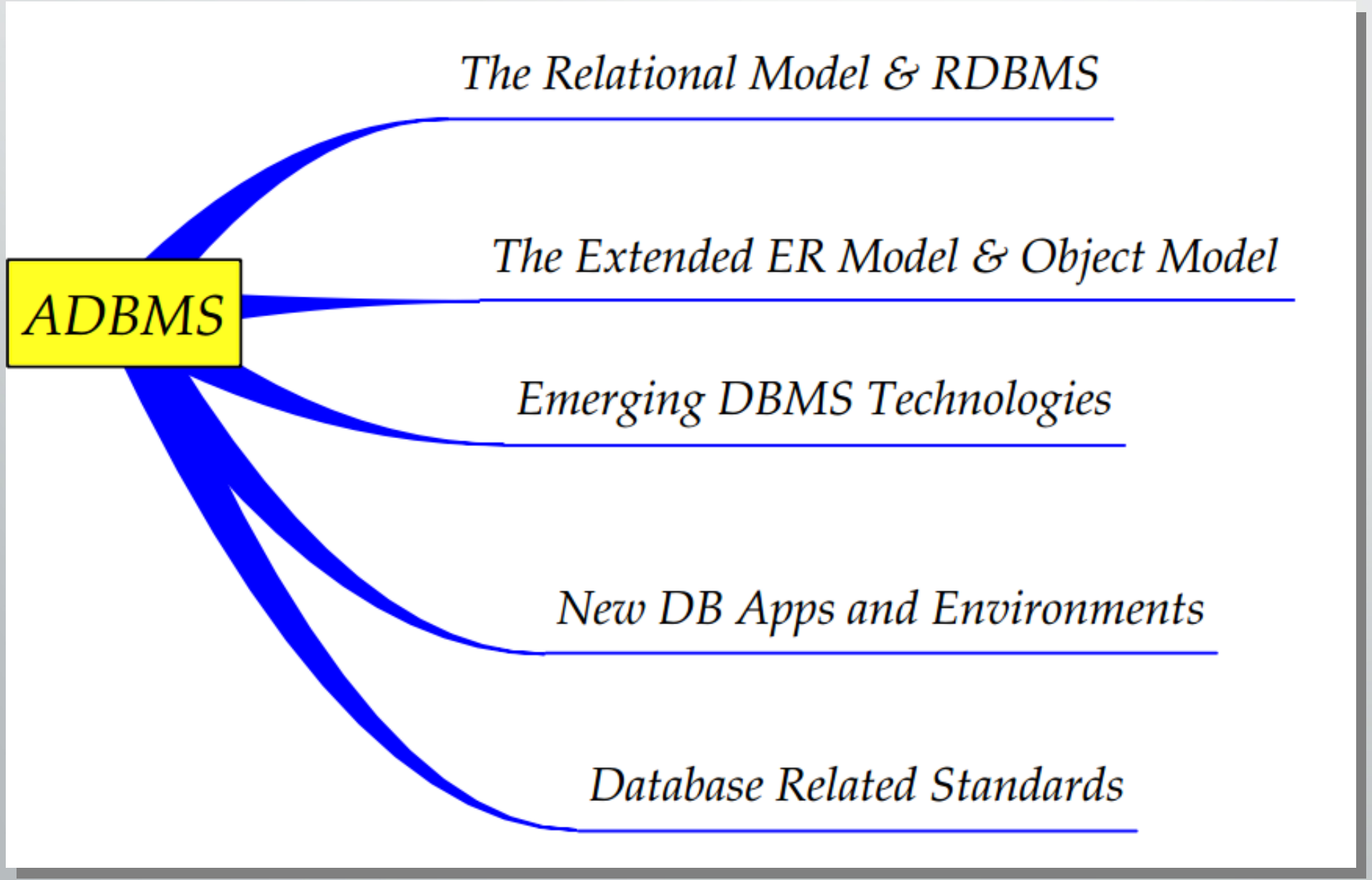
Advanced
Database Management System

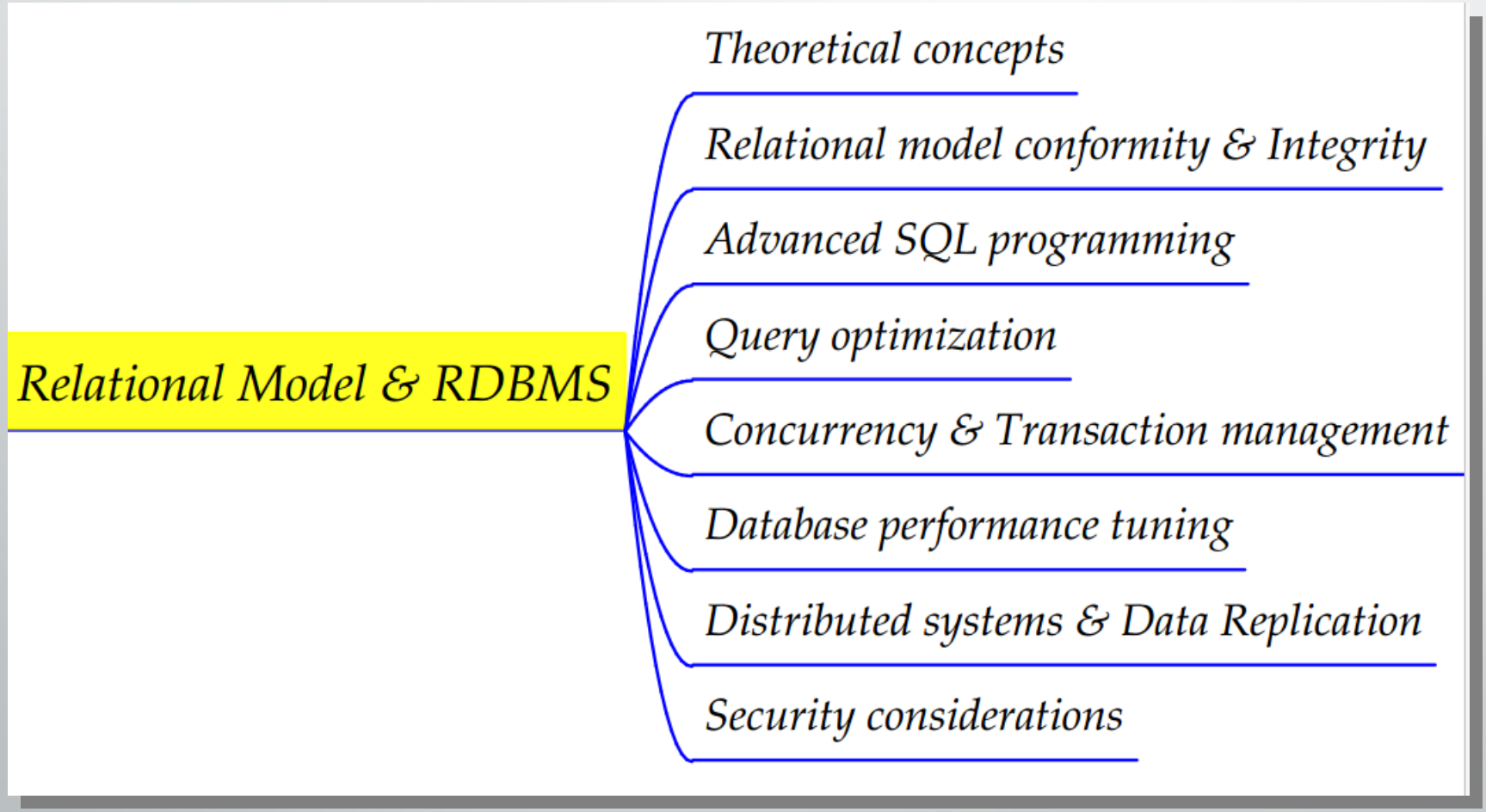
[BSc CSIT-7th Semester]

Rishi K. Marseni
{rishimarseni@gmail.com}

Course Content – Abstract View



The Relational Model of Data and RDBMS Implementation Techniques



Why ADBMS?

- Controlling Redundancy
- Restricting Unauthorized Access
- Providing Persistent Storage for Program Objects and Data Structures
- Permitting Inferencing and Actions Using Rules
- Providing Multiple User Interfaces
- Representing Complex Relationships Among Data
- Enforcing Integrity Constraints
- Providing Backup and Recovery

Concurrency control and Transaction management

- Database transaction and the ACID rules
- Why is concurrency control needed?
- Concurrency control mechanisms
  - → Categories
  - → Methods
  - → Major goals
- Transaction Processing
  - → Rollback / Roll forward / deadlocks

Database transaction and the ACID rules

- The concept of a database transaction (or atomic transaction) has evolved in order to enable both a well understood database system behavior in a **faulty environment** where **crashes** can happen any time, and recovery from a crash to a well understood database state.
- A database transaction is a unit of work, typically **encapsulating** a number of operations over a database (e.g., reading a database object, writing, acquiring lock, etc.), an abstraction supported in database and also other systems.
- Each transaction has well defined **boundaries** in terms of which program/code executions are included in that transaction (determined by the transaction's programmer via special transaction commands).
- Every database transaction obeys the following rules (by support in the database system; i.e., a database system is designed to guarantee them for the transactions it runs):

ACID Properties

Atomicity:

- Either the effects of all or none of its operations remain ("all or nothing" semantics) when a transaction is completed (committed or aborted respectively).
- In other words, to the outside world a committed transaction appears (by its effects on the database) to be indivisible, atomic, and an aborted transaction does not leave effects on the database at all, as if never existed

ACID Properties

Consistency

- Every transaction must leave the database in a consistent (correct) state, i.e., maintain the predetermined integrity rules of the database (constraints upon and among the database's objects).
- A transaction must transform a database from one consistent state to another consistent state
- However, it is the responsibility of the transaction's programmer to make sure that the transaction itself is correct
- Performs correctly what it intends to perform (from the application's point of view) while the predefined integrity rules are enforced by the DBMS).
- Thus since a database can be normally changed only by transactions, all the database's states are consistent.
- An aborted transaction does not change the database state it has started from, as if it never existed (atomicity above).

ACID Properties

Isolation

- Transactions cannot interfere with each other (as an end result of their executions).
- Moreover, usually (depending on concurrency control method) the effects of an incomplete transaction are not even visible to another transaction.
- Providing isolation is the main goal of concurrency control.

Durability

- Effects of successful (committed) transactions must persist through crashes (typically by recording the transaction's effects and its commit event in a non-volatile memory).

Why is concurrency control needed?

If transactions are executed serially, i.e., sequentially with no overlap in time, no transaction concurrency exists.
However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur. Here are some typical examples:

The lost update problem:

- A second transaction writes a second value of a data-item (datum) on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value.
- The transactions that have read the wrong value end with incorrect results.

The dirty read problem:

- Transactions read a value written by a transaction that has been later aborted.
- This value disappears from the database upon abort, and should not have been read by any transaction
- ("dirty read"). The reading transactions end with incorrect results.

Why is concurrency control needed?

The incorrect summary problem:

- While one transaction takes a summary over the values of all the instances of a repeated data-item, a second transaction updates some instances of that data-item.
- The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

Most high-performance transactional systems need to run transactions concurrently to meet their performance requirements.
Thus, without concurrency control such systems can neither provide correct results nor maintain their databases consistent.

Concurrency control mechanisms

Categories

The main categories of concurrency control mechanisms are:

- **Optimistic:**
- → Delay the checking of whether a transaction meets the isolation and other integrity rules (e.g., **serializability** and **recoverability**) until its end, without blocking any of its (read, write) operations ("...and be optimistic about the rules being met..."), and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit.
- → An aborted transaction is immediately restarted and re-executed, which incurs an obvious overhead (versus executing it to the end only once).
- → If not too many transactions are aborted, then being optimistic is usually a good strategy.

Concurrency control mechanisms

Pessimistic:

→ Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears.

→ Blocking operations is typically involved with performance reduction.

Semi-Pessimistic:

→ Block operations in some situations, if they may cause violation of some rules, and do not block in other situations while delaying rules checking (if needed) to transaction's end, as done with optimistic.

Different categories provide different performance, i.e., different average transaction completion rate (throughput), depending on transaction types mix, computing level of parallelism, and other factors.

If selection and knowledge about trade-offs are available, then category and method should be chosen to provide the highest performance.

Concurrency Control Methods

- Many methods for concurrency control exist. Most of them can be implemented within either main category above.
- The major methods, which have each many variants, and in some cases may overlap or be combined, are:

  -> Locking (e.g., Two-phase locking - 2PL)

  -> Serialization graph checking (Precedence graph checking)

  -> Timestamp ordering (TO)

  -> Commitment ordering (or Commit ordering; CO)

Methods

- Many methods for concurrency control exist. Most of them can be implemented within either main category above.
- There are main three methods for concurrency control. They are as follows:
    1. Locking Methods
    2. Time-stamp Methods
    3. Optimistic Methods

Locking Methods of Concurrency Control

- "A lock is a variable, associated with the data item, which controls the access of that data item."
- Locking is the most widely used form of the concurrency control.
- Locks are further divided into three fields:
  - → Lock Granularity
  - → Lock Types
  - → Deadlocks

Locking Methods of Concurrency Control

1. Lock Granularity :

- A database is basically represented as a collection of named data items.
- The size of the data item chosen as the unit of protection by a concurrency control program is called GRANULARITY.
- Locking can take place at the following level :

  -> Database level: the entire database is locked

  -> Table level: the entire table is locked

  -> Page level: the entire disk-page (or disk-block) is locked

  -> Row (Tuple) level: particular row (or tuple) is locked

  -> Attributes (fields) level: particular attribute (or field) is locked

Locking Methods of Concurrency Control

2. Lock Types :

The DBMS mainly uses following types of locking techniques.

Binary Locking:

A binary lock can have two states or values:

locked and unlocked (or 1 and 0, for simplicity).

A distinct lock is associated with each database item X using LOCK(X)

Operations: Lock_item(X) and Unlock_item(X)

Shared / Exclusive Locking:

Shared locks are referred as read locks, and denoted by 'S'.

Exclusive locks are referred as Write locks, and denoted by 'X'.

Two - Phase Locking (2PL):

Two-phase locking (also called 2PL) is a method or a protocol of controlling concurrent processing in which all locking operations precede the first unlocking operation.

Locking Methods of Concurrency Control

More about 2PL:
A transaction is said to follow the two-phase locking protocol if all locking operations (such as read_Lock, write_Lock) precede the first unlock operation in the transaction.
Two-phase locking is the standard protocol used to maintain level 3 consistency
2PL defines how transactions acquire and relinquish locks.
The essential discipline is that after a transaction has released a lock it may not obtain any further locks.

2PL has the following two phases:
A **growing** phase, in which a transaction acquires all the required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked point.
A **shrinking** phase, in which a transaction releases all locks and cannot obtain any new lock.

Locking Methods of Concurrency Control

Time	Transaction	Remarks
t0	Lock - X (A)	acquire Exclusive lock on A.
t1	Read A	read original value of A
t2	A = A - 100	subtract 100 from A
t3	Write A	write new value of A
t4	Lock - X (B)	acquire Exclusive lock on B.
t5	Read B	read original value of B
t6	B = B + 100	add 100 to B
t7	Write B	write new value of B
t8	Unlock (A)	release lock on A
t9	Unock (B)	release lock on B

Figure: 2PL Example

Locking Methods of Concurrency Control

3. Deadlocks:

- A deadlock is a condition in which two (or more) transactions in a set are waiting simultaneously for locks held by some other transaction in the set.
- Neither transaction can continue because each transaction in the set is on a waiting queue, waiting for one of the other transactions in the set to release the lock on an item.
- Thus, a deadlock is an impasse that may result when two or more transactions are each waiting for locks to be released that are held by the other.
- Transactions whose lock requests have been refused are queued until the lock can be granted.
  A deadlock is also called a circular waiting condition where two transactions are waiting (directly or indirectly) for each other.

- Thus in a deadlock, two transactions are mutually excluded from accessing the next record required to complete their transactions, also called a deadly embrace.

Locking Methods of Concurrency Control

Deadlock Example:

A deadlock exists two transactions A and B exist in the following example:

Transaction A = access data items X and Y

Transaction B = access data items Y and X

Here, Transaction-A has acquired lock on X and is waiting to acquire lock on y.
While, Transaction-B has acquired lock on Y and is waiting to acquire lock on X.
But, none of them can execute further.

Transaction-A	Time	Transaction-B
Lock (X) (acquired lock on X)	t1	---
---	t2	Lock (Y) (acquired lock on Y)
Lock (Y) (request lock on Y)	t3	---
Wait	t4	Lock (X) (request lock on X)
Wait	t5	Wait

Timestamp Methods for Concurrency Control

Timestamp is a unique identifier created by the DBMS to identify the relative starting time of a transaction.
Timestamp values are assigned in the order in which the transactions are submitted to the system.
A timestamp can be thought of as the transaction start time, time stamping is a method of concurrency control in which each transaction is assigned a transaction timestamp.
Uniqueness :
The uniqueness property assures that no equal timestamp values can exist.
Monotonicity : monotonicity assures that timestamp values always increase.
Timestamp are divided into further fields :
Granule Timestamps
Timestamp Ordering
Conflict Resolution in Timestamps

Timestamp Methods for Concurrency Control

1. Granule Timestamps :

- Granule timestamp is a record of the timestamp of the last transaction to access it.
- Each granule accessed by an active transaction must have a granule timestamp.
  A separate record of last Read and Write accesses may be kept.
- Granule timestamp may cause additional Write operations for Read accesses if they are stored with the granules.
- The problem can be avoided by maintaining granule timestamps as an in-memory table.
- The table may be of limited size, since conflicts may only occur between current transactions.
- An entry in a granule timestamp table consists of the granule identifier and the transaction timestamp.
- The record containing the largest (latest) granule timestamp removed from the table is also maintained.
- A search for a granule timestamp, using the granule identifier, will either be successful or will use the largest removed timestamp.

Timestamp Methods for Concurrency Control

2. Timestamp Ordering :

Following are the three basic variants of timestamp-based methods of concurrency control:

- **Total timestamp ordering**

  No distinction is made between Read and Write access, so only a single value is required for each granule timestamp .

- **Partial timestamp ordering**

  Only non-permutable actions are ordered to improve upon the total timestamp ordering. Both Read and Write granule timestamps are stored.

- **Multiversion timestamp ordering**

  The multiversion timestamp ordering algorithm stores several versions of an updated granule, allowing transactions to see a consistent set of versions for all granules it accesses.

Timestamp Methods for Concurrency Control

3. Conflict Resolution in Timestamps :

To deal with conflicts in timestamp algorithms, some transactions involved in conflicts are made to wait and to abort others.

Following are the main strategies of conflict resolution in timestamps:

WAIT-DIE: The older transaction waits for the younger if the younger has accessed the granule first.

The younger transaction is aborted (dies) and restarted if it tries to access a granule after an older concurrent transaction.

WOUND-WAIT: The older transaction preempts the younger by suspending (wounding) it if the younger transaction tries to access a granule after an older concurrent transaction.

An older transaction will wait for a younger one to commit if the younger has accessed a granule that both want.

Timestamp Methods for Concurrency Control

The handling of aborted transactions is an important aspect of conflict resolution algorithm. In the case that the aborted transaction is the one requesting access, the transaction must be restarted with a new (younger) timestamp. It is possible that the transaction can be repeatedly aborted if there are conflicts with other transactions.

An aborted transaction that had prior access to granule where conflict occurred can be restarted with the same timestamp. This will take priority by eliminating the possibility of transaction being continuously locked out.

Drawbacks of Time-stamp:

Each value stored in the database requires two additional timestamp fields, one for the last time the field (attribute) was read and one for the last update.

It increases the memory requirements and the processing overhead of database.

Optimistic Methods of Concurrency Control

The optimistic method of concurrency control is based on the assumption that conflicts of database operations are rare and that it is better to let transactions run to completion and only check for conflicts before they commit.
An optimistic concurrency control method is also known as validation or certification methods.
No checking is done while the transaction is executing.
The optimistic method does not require locking or timestamping techniques.
Instead, a transaction is executed without restrictions until it is committed.
In optimistic methods, each transaction moves through the following phases:

- Read phase.
- Validation or certification phase.
- Write phase.

Optimistic Methods of Concurrency Control

Advantages of Optimistic Methods for Concurrency Control :
This technique is very efficient when conflicts are rare.
The occasional conflicts result in the transaction roll back.
The rollback involves only the local copy of data, the database is not involved and thus there will not be any cascading rollbacks.
Problems of Optimistic Methods for Concurrency Control :
Conflicts are expensive to deal with, since the conflicting transaction must be rolled back.
Longer transactions are more likely to have conflicts and may be repeatedly rolled back because of conflicts with short transactions.
Applications of Optimistic Methods for Concurrency Control :
Only suitable for environments where there are few conflicts and no long transactions.
Acceptable for mostly Read or Query database systems that require very few update transactions

Concurrency Control Method

Other major concurrency control types that are utilized in conjunction with the methods above include:

- **Multiversion concurrency control** (MVCC) - Increasing concurrency and performance by generating a new version of a database object each time the object is written, and allowing transactions' read operations of several last relevant versions (of each object) depending on scheduling method.
- **Index concurrency control** - Synchronizing access operations to indexes, rather than to user data. Specialized methods provide substantial performance gains.
- **Private workspace model** (**Deferred update**) - Each transaction maintains a private workspace for its accessed data, and its changed data become visible outside the transaction only upon its commit(e.g., Weikum and Vossen 2001). This model provides a different concurrency control behavior with benefits in many cases.

Goals of Concurrency Control

Concurrency control mechanisms firstly need to operate correctly, i.e., to maintain each transaction's integrity rules (as related to concurrency; application-specific integrity rule are out of the scope here) while transactions are running concurrently, and thus the integrity of the entire transactional system.

Correctness needs to be achieved with as good performance as possible. In addition, increasingly a need exists to operate effectively while transactions are distributed over processes, computers, and computer networks.

Other subjects that may affect concurrency control are recovery and replication.
Correctness [*Serializability, Recoverability, Replicable***]**

Transaction Processing

Transaction processing is information processing that is divided into individual, indivisible operations, called *transactions.*

Each transaction must succeed or fail as a complete unit; it cannot remain in an intermediate state.

Since most, though not necessarily all, transaction processing today is interactive the term is often treated as synonymous with **online transaction processing.**

The basic principles of all transaction-processing systems are the same. However, the terminology may vary from one transaction-processing system to another, and the terms used below are not necessarily universal.

Rollback

Rollforward

Compensating transaction:

In systems where commit and rollback mechanisms are not available or undesirable, a compensating transaction is often used to undo failed transactions and restore the system to a previous state.

Transaction Processing

Transaction processing has these benefits:

- It allows sharing of computer resources among many users
- It shifts the time of job processing to when the computing resources are less busy
- It avoids idling the computing resources without minute-by-minute human interaction and supervision
- It is used on expensive classes of computers to help amortize the cost by keeping high rates of utilization of those expensive resources

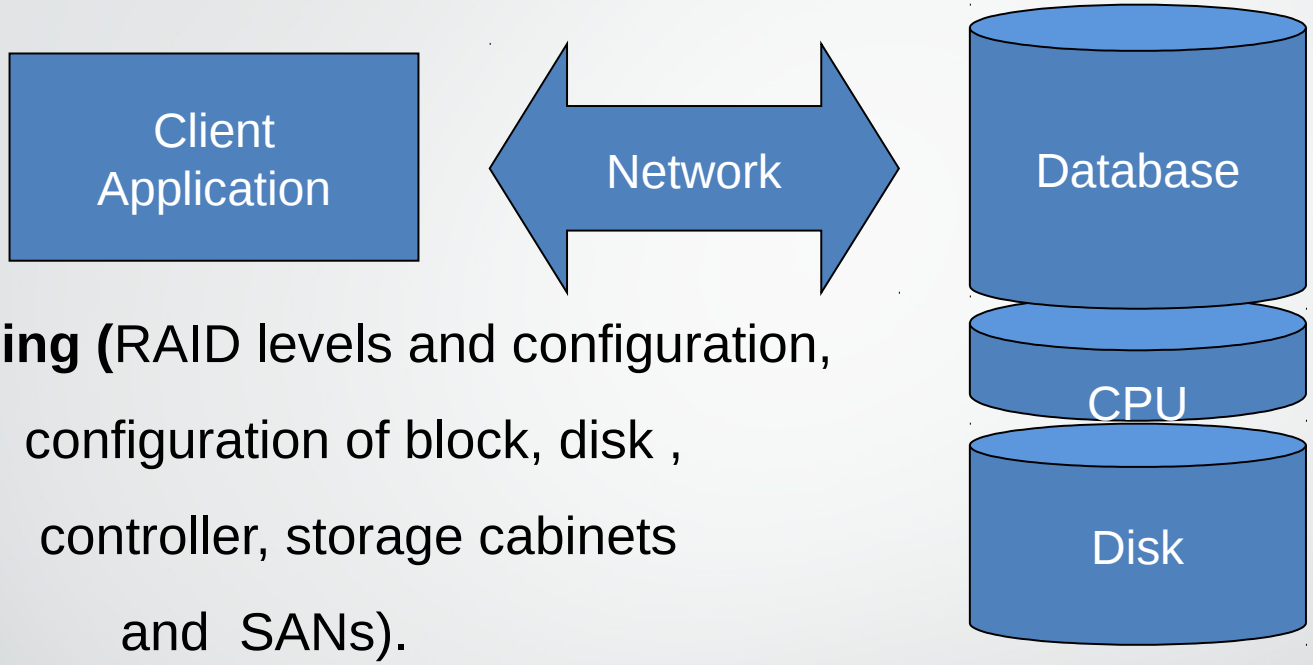
Database Performance Tuning

- Trade-offs Between Response Time and Throughput
- Throughput is measured in blocks per second and response time is measured in transaction per second.
- Goals for tuning vary, depending on the needs of the application
- OLTP(Transaction) vs. OLAP(Analytical) applications (which one requires which performance?)
- Response time = service time + wait time
- We can increase performance two ways:
  by reducing service time
  by reducing wait time

Database Performance Tuning

- **Database tuning** describes a group of activities used to optimize and homogenize the performance of a database.
- It usually overlaps with query tuning, but refers to design of the database files, selection of the database management system (DBMS), operating system and CPU the DBMS runs on.
- Database tuning aims to maximize use of system resources to perform work as efficiently and rapidly as possible.
- Most systems are designed to manage their use of system resources, but there is still much room to improve their efficiency by customizing their settings and configuration for the database and the DBMS.

Database Performance Tuning

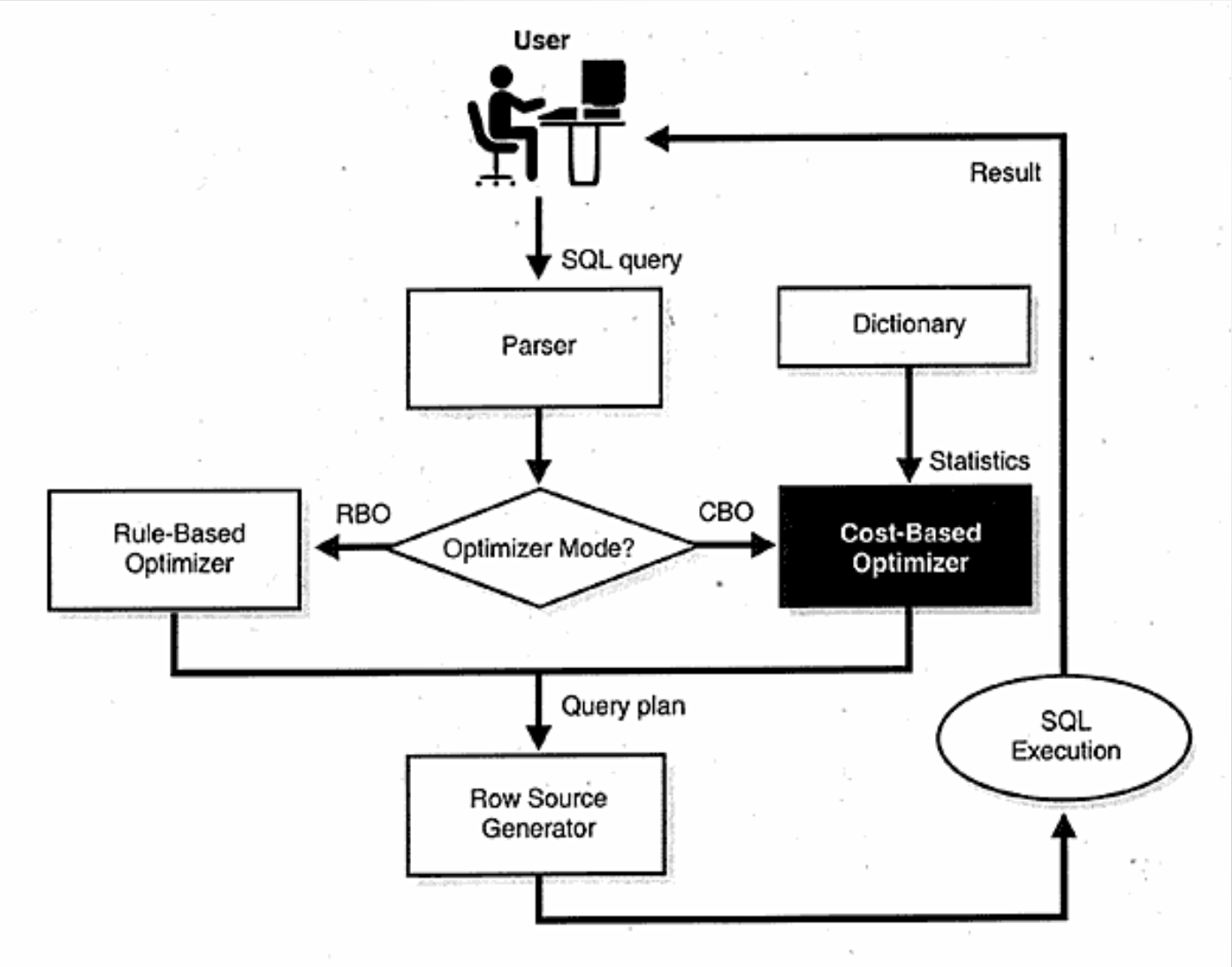


1. **I/O tuning (**RAID levels and configuration, configuration of block, disk , controller, storage cabinets and SANs).

2. DBMS tuning(configuration of memory and processing resources)

3. Database maintenance(backups, column statistics updates, and defragmentation of data inside the database files.)

Database Performance Tuning



Database Performance Tuning

Goal of database performance is to execute queries as fast as possible

Database performance tuning: Set of activities and procedures that reduce response time of database system

Fine-tuning the performance of a system requires that all factors must operate at optimum level with minimal bottlenecks

Client side(**SQL performance tuning)**:

Generates SQL query that returns correct answer in least amount of time

Using minimum amount of resources at server

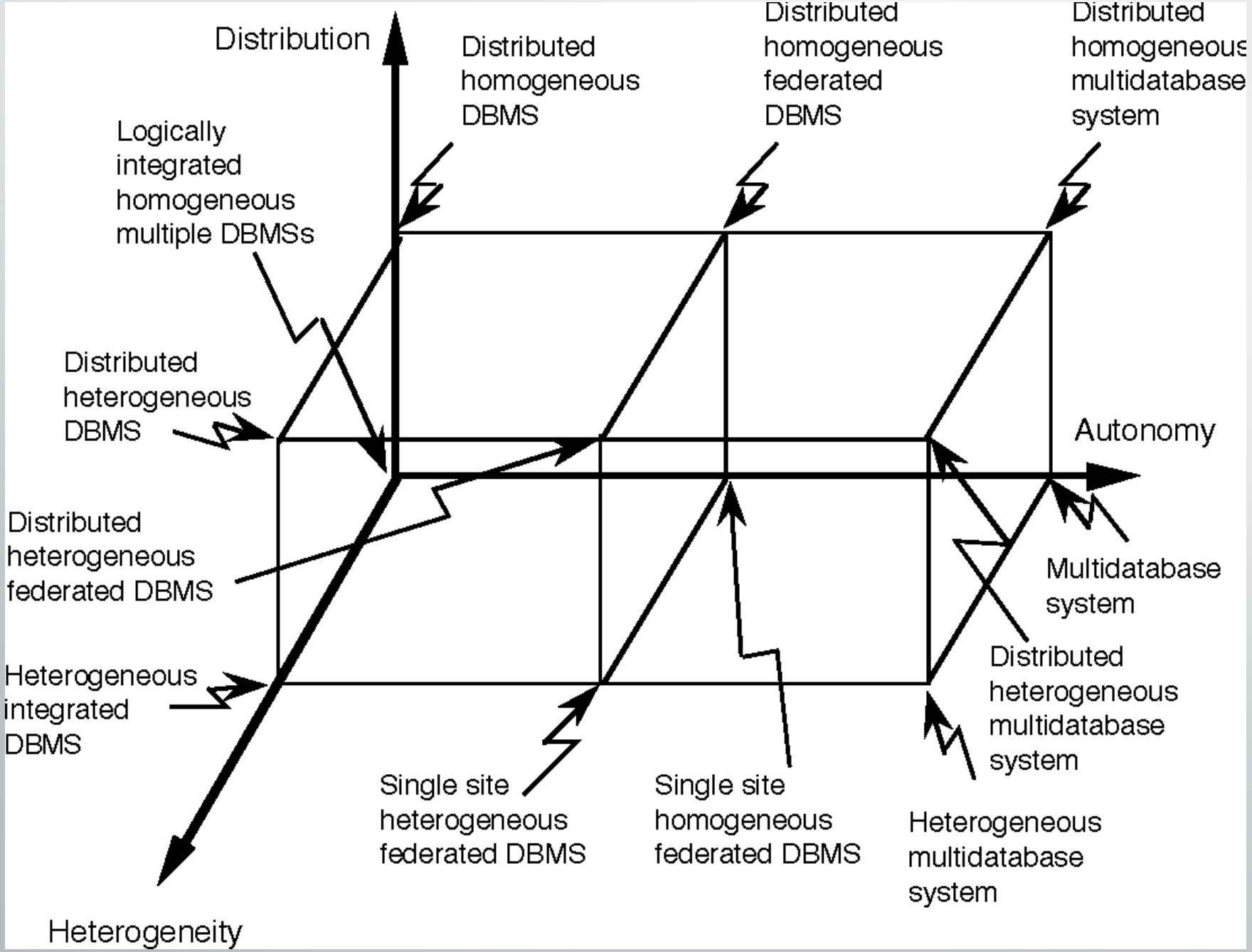
Server side(**DBMS performance tuning):** DBMS environment configured to respond to clients' requests as fast as possible

Optimum use of existing resources

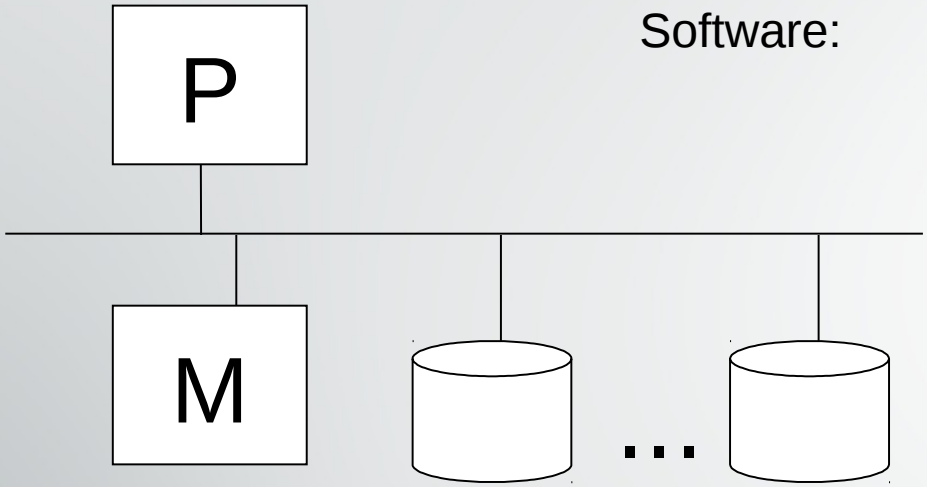
General Performance Tuning Guideline

	SYSTEM RESOURCES	CLIENT	SERVER
Hardware	CPU	The fastest possible Dual-core CPU or higher	The fastest possible Multiple processors (quad-core technology) Cluster of networked computers
	RAM	The maximum possible to avoid OS memory to disk swapping	The maximum possible to avoid OS memory to disk swapping
	Hard disk	Fast SATA/EIDE hard disk with sufficient free hard disk space Solid State Drives (SSD) for faster speed	Multiple high-speed, high-capacity disks Fast disk interface (SAS / SCSI / Firewire / Fibre Channel RAID configuration optimized for throughput Solid State Drives (SSD) for faster speed Separate disks for OS, DBMS, and data spaces
	Network	High-speed connection	High-speed connection
Software	Operating System (OS)	64-bit OS for larger address spaces Fine-tuned for best client application performance	64-bit OS for larger address spaces Fine-tuned for best server application performance
	Network	Fine-tuned for best throughput	Fine-tuned for best throughput
	Application	Optimize SQL in client application	Optimize DBMS server for best performance

Distributed System and Data Replication



Centralized DB systems



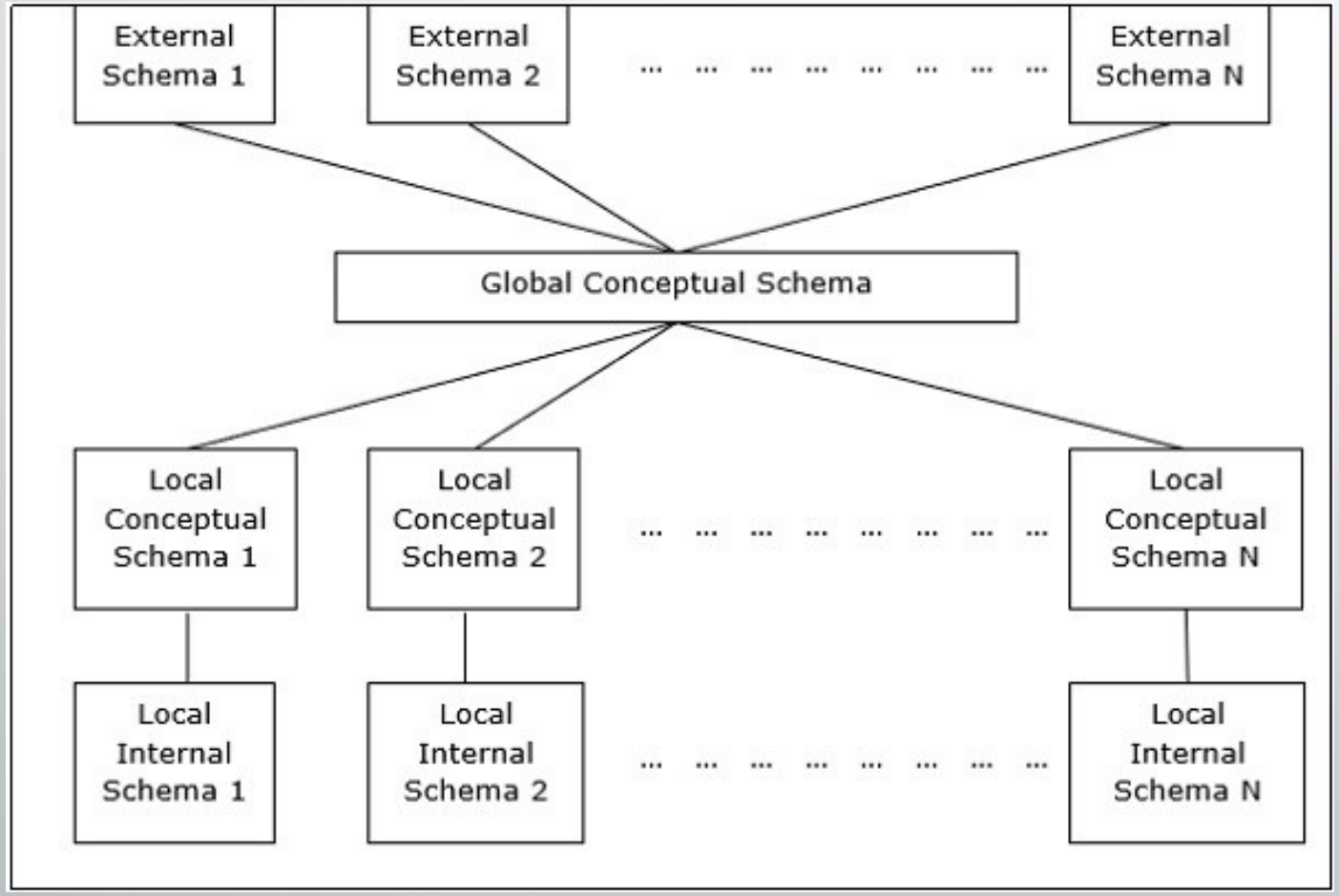
Software:

Application
SQL Front End
Query Processor
Transaction Proc.
File Access

- Simplifications:
  - single front end
  - one place to keep data, locks
  - if processor fails, system fails, ...

Distributed Database Systems

Multiple processors (+ memories)
Heterogeneity and autonomy of “components”



Why do we need Distributed Databases?

Example: IBM has offices in London, New York, and Hong Kong.
Employee data:
EMP(ENO, NAME, TITLE, SALARY, …)
Where should the employee data table reside?

IBM Data Access Pattern

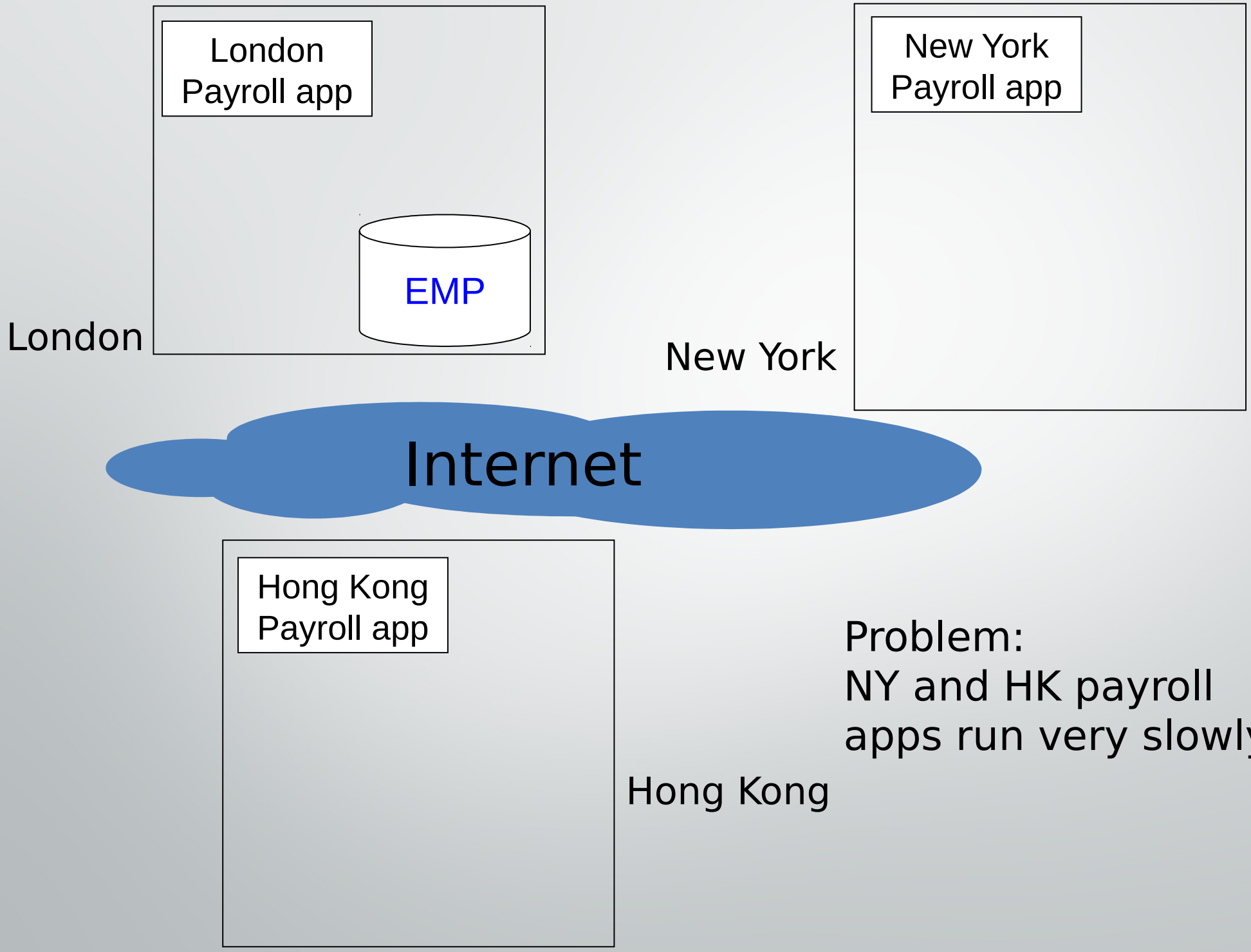
Mostly, employee data is managed at the office where the employee works

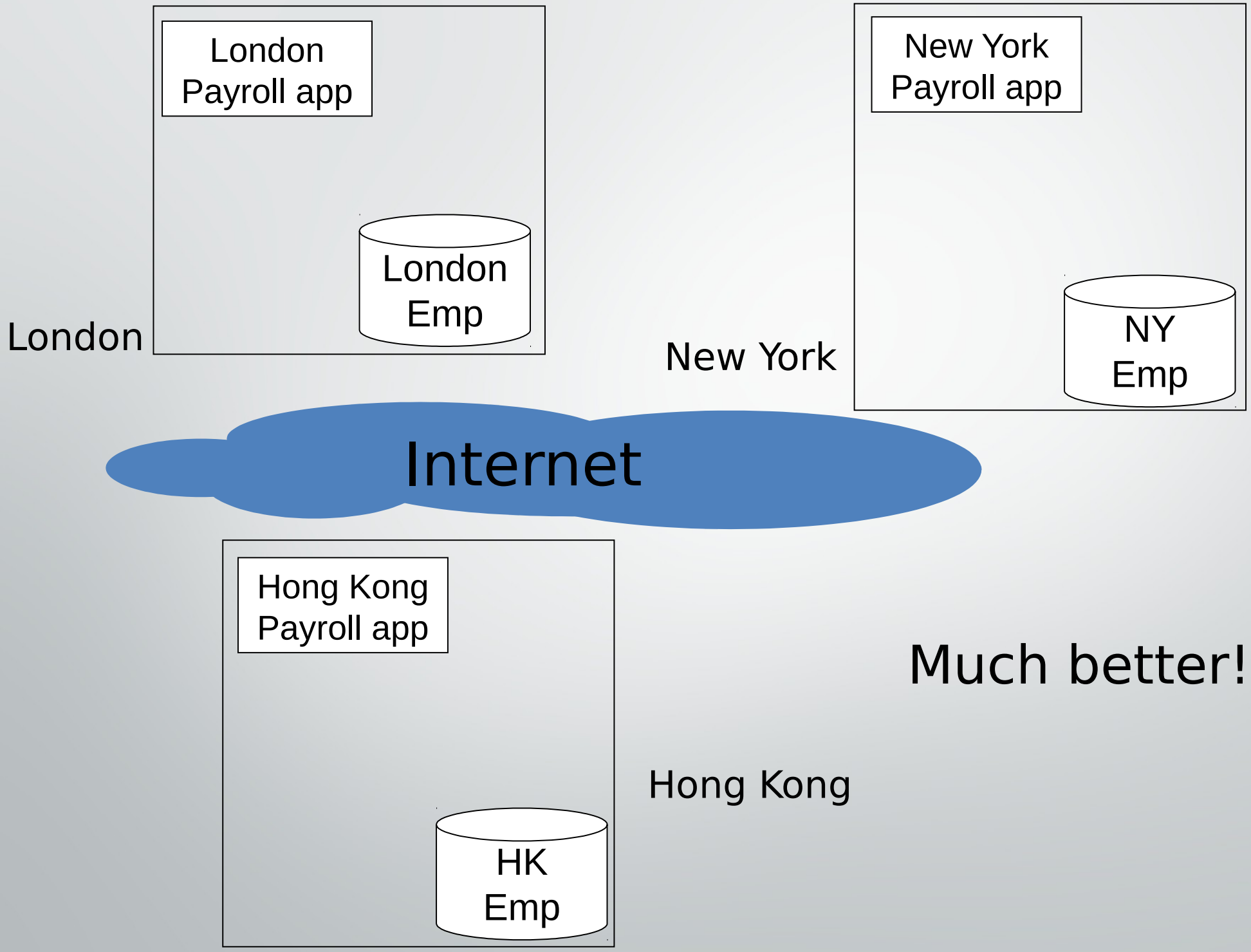
E.g., payroll, benefits, hire and fire

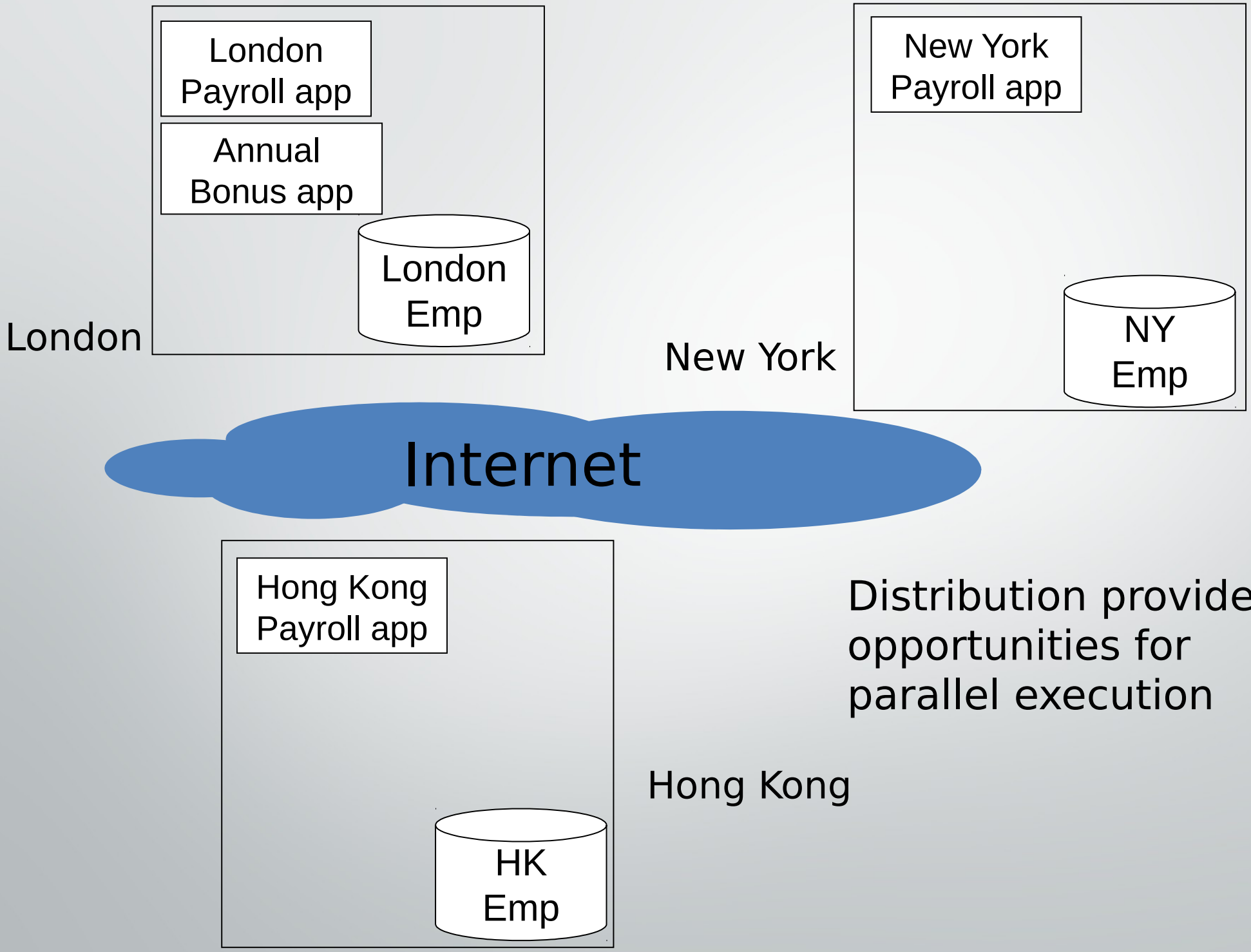
Periodically, IBM needs consolidated access to employee data

E.g., IBM changes benefit plans and that affects all employees.

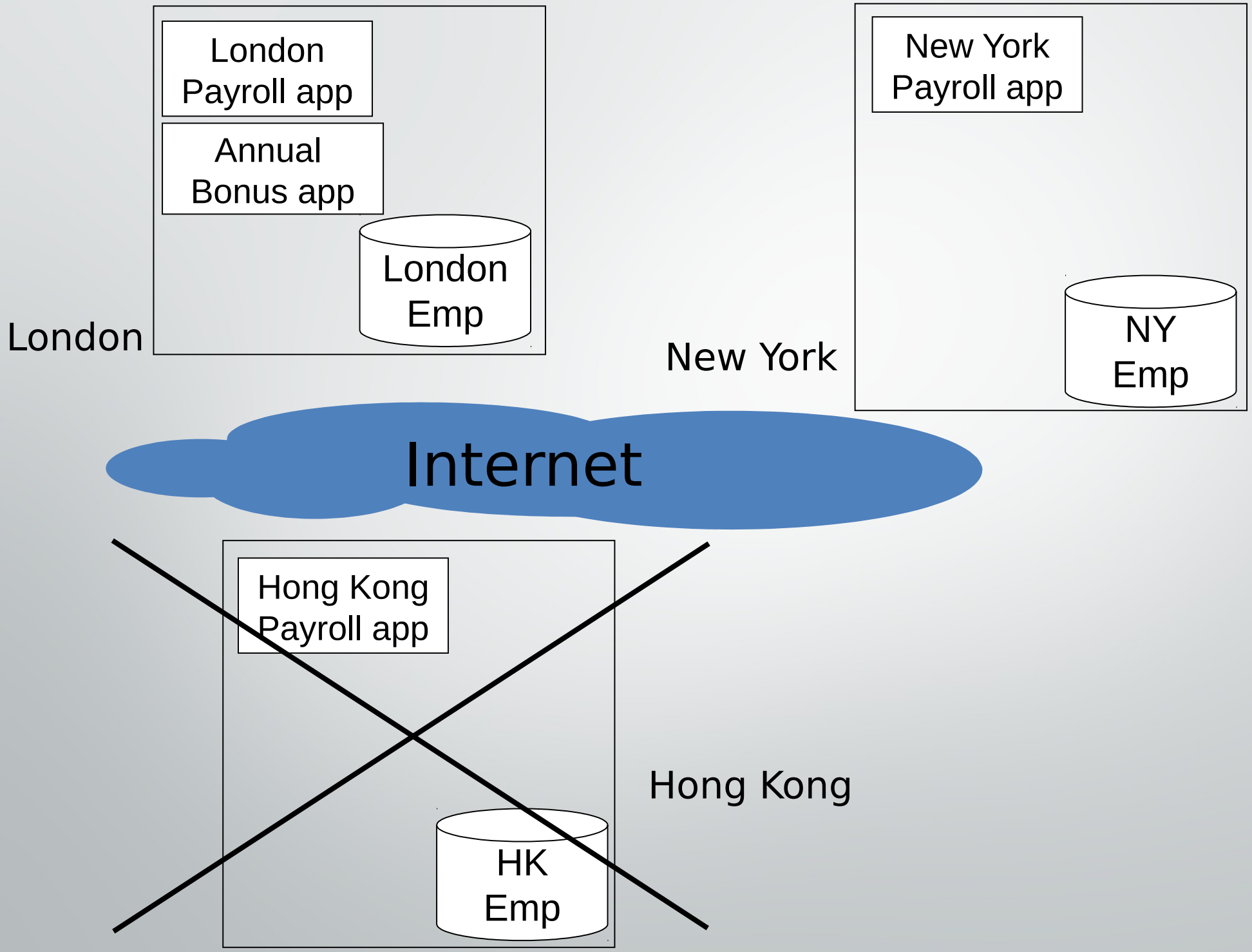
E.g., Annual bonus depends on global net profit.

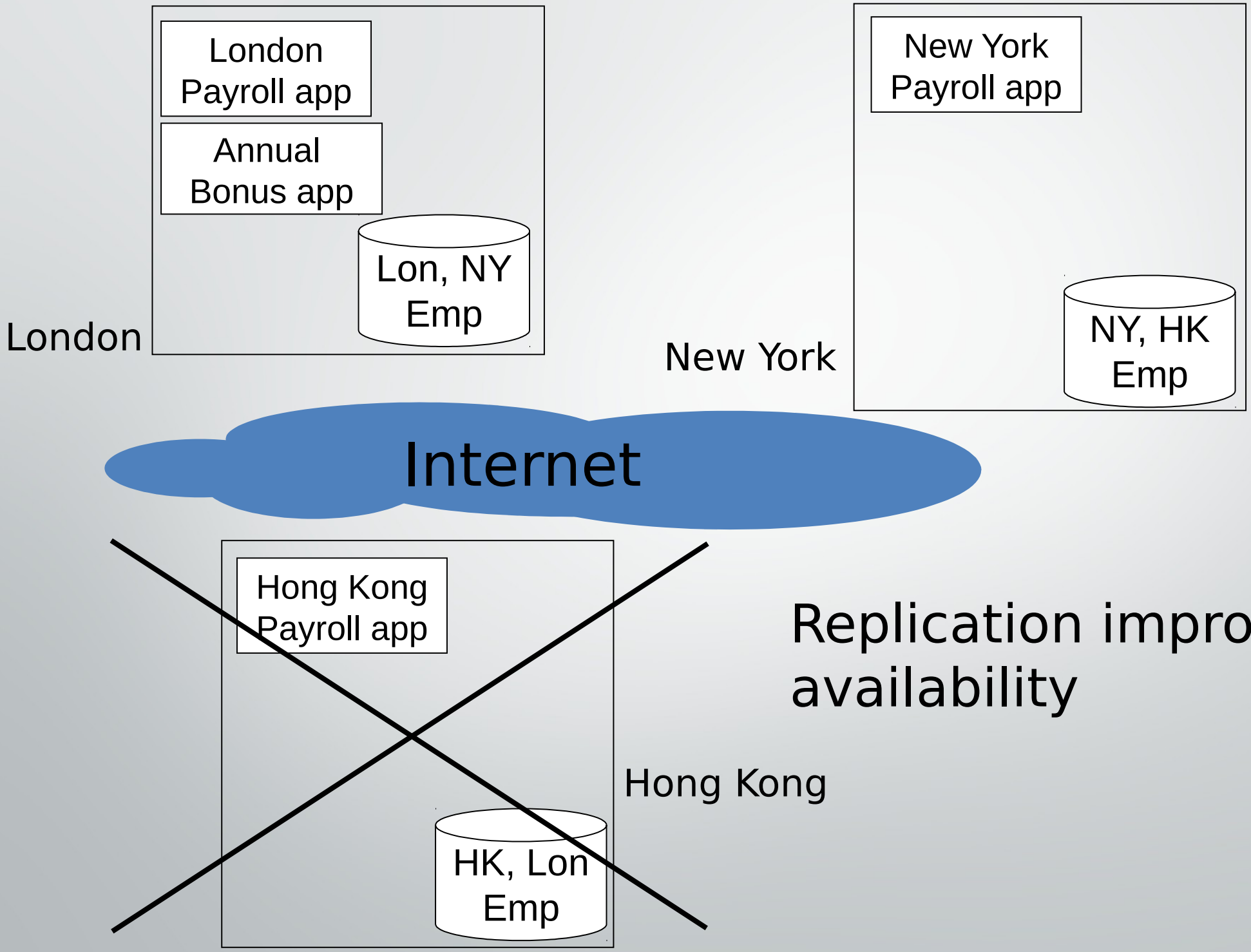

London
Payroll app
EMP
London
New York
Payroll app
New York
Internet
Hong Kong
Payroll app
Hong Kong
Problem:
NY and HK payroll
apps run very slowly!


London
Payroll app
London
Emp
London
New York
Payroll app
NY
Emp
New York
Internet
Hong Kong
Payroll app
HK
Emp
Hong Kong
Much better!!



Distribution provides opportunities for parallel execution


London Payroll app
Annual Bonus app
London Emp
London
New York Payroll app
NY Emp
New York
Internet
Hong Kong Payroll app
HK Emp
Hong Kong


London Payroll app
Annual Bonus app
Lon, NY Emp
London
New York Payroll app
NY, HK Emp
New York
Internet
Hong Kong Payroll app
HK, Lon Emp
Hong Kong
Replication improves availability

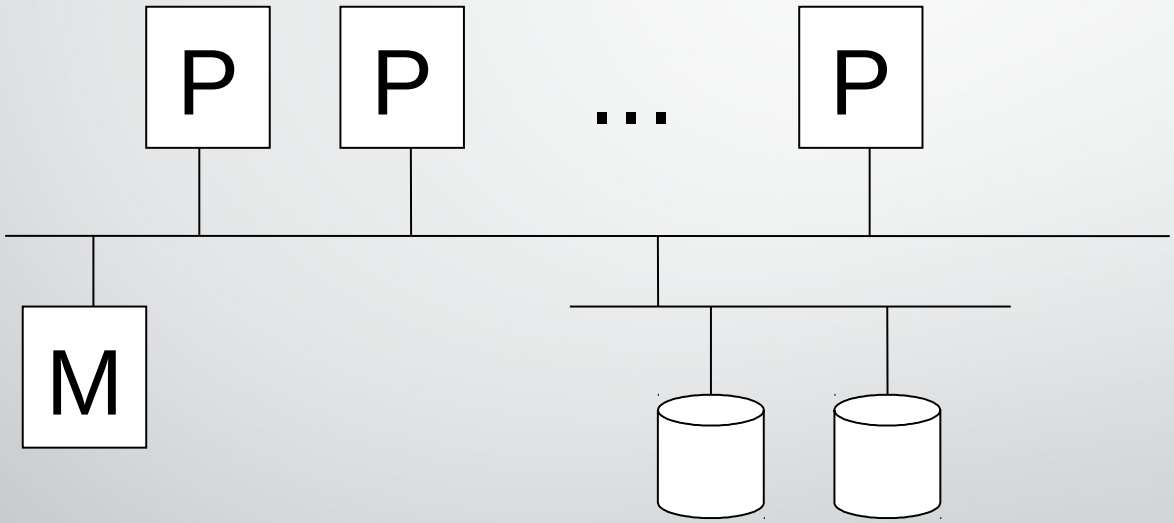
Homogeneous Distributed Databases

In a 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

In a 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

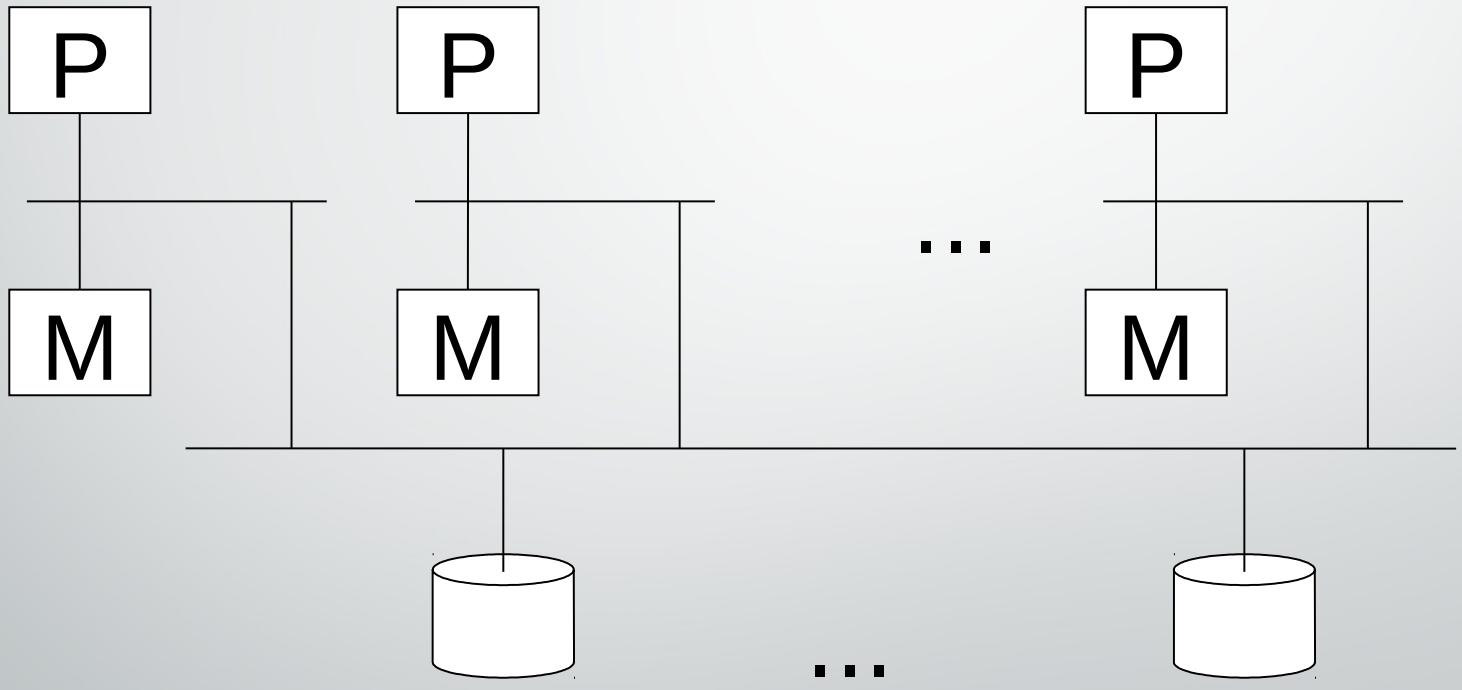
DB Architectures

(1) Shared memory



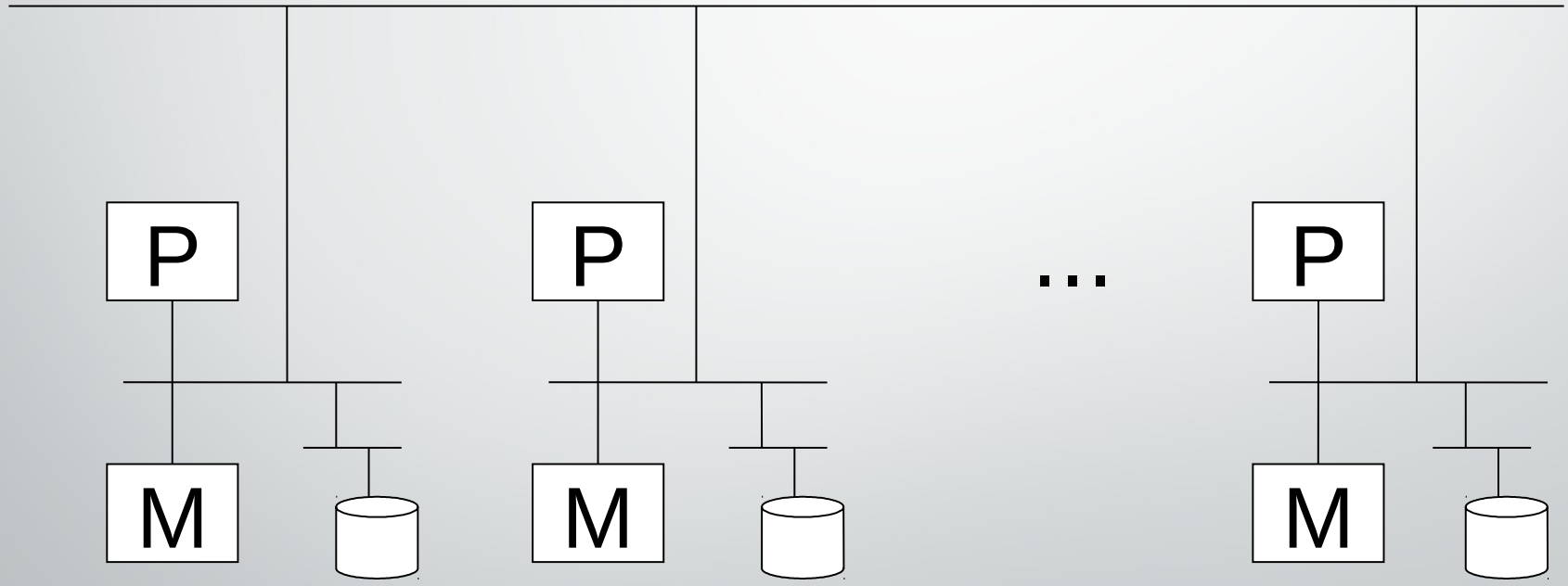
DB Architectures

(2) Shared disk



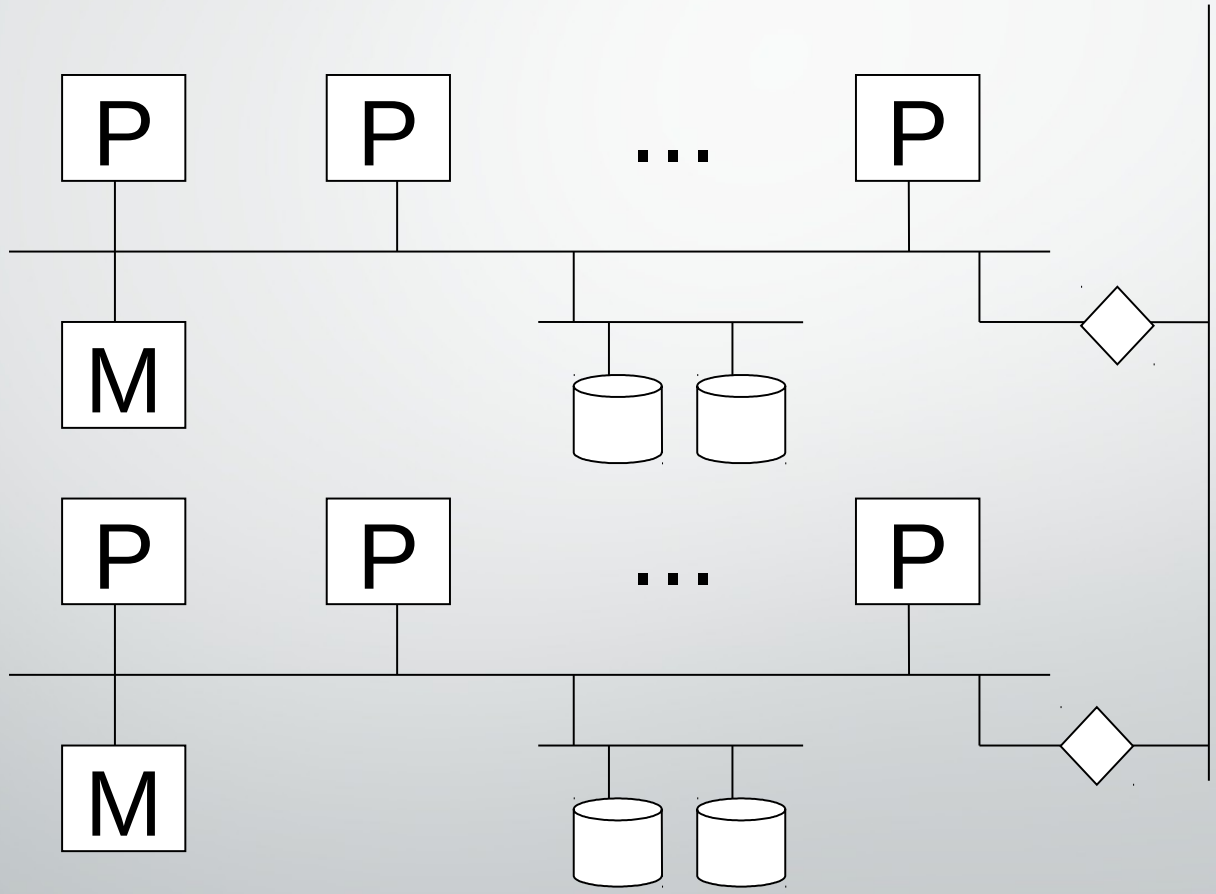
DB Architectures

(3) Shared nothing



DB Architectures

(4) Hybrid example – Hierarchical or Clustered



Issues for selecting architecture

- Reliability
- Scalability
- Geographic distribution of data
- Performance
- Cost

Parallel or distributed DB system?

More similarities than differences!

Typically, parallel DBs:

- Fast interconnect
- Homogeneous software
- High performance is goal
- Transparency is goal

Typically, distributed DBs:

- Geographically distributed
- Data sharing is goal (may run into heterogeneity, autonomy)
- Disconnected operation possible

Distributed Database Challenges

- Distributed Database Design
- Deciding what data goes where
- Depends on data access patterns of major applications
- Two sub-problems:
  -> Fragmentation: partition tables into fragments
  -> Allocation: allocate fragments to nodes

Distributed Data Storage

Assume relational data model

Replication

System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.

Fragmentation

Relation is partitioned into several fragments stored in distinct sites

Replication and fragmentation can be combined

Relation is partitioned into several fragments: system maintains several identical replicas of each such fragment.

Data Replication

- A relation or fragment of a relation is **replicated** if it is stored redundantly in two or more sites.
- Full replication of a relation is the case where the relation is stored at all sites.
- Fully redundant databases are those in which every site contains a copy of the entire database.

Data Replication (Cont.)

[Advantages of Replication]
Availability: failure of site containing relation *r* does not result in unavailability of *r* is replicas exist.
Parallelism: queries on *r* may be processed by several nodes in parallel.
Reduced data transfer: relation *r* is available locally at each site containing a replica of *r*.

[Disadvantages of Replication]
Increased cost of updates: each replica of relation *r* must be updated.
Increased complexity of concurrency control: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.
One solution: choose one copy as **primary copy** and apply concurrency control operations on primary copy

Data Fragmentation

Division of relation r into fragments r_1, r_2, …, r_n which contain sufficient information to reconstruct relation r.

Horizontal fragmentation: each tuple of *r* is assigned to one or more fragments

Vertical fragmentation: the schema for relation *r* is split into several smaller schemas

All schemas must contain a common candidate key (or superkey) to ensure lossless join property.

A special attribute, the tuple-id attribute may be added to each schema to serve as a candidate key.

Example : relation account with following schema

Account = (*branch_name*, *account_number, balance*)

Horizontal Fragmentation of *account* Relation

branch_name	*account_number*	*balance*
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

$$account_1 = \sigma_{branch\_name=\text{“Hillside”}}(account)$$

branch_name	*account_number*	*balance*
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

$$account_2 = \sigma_{branch\_name=\text{“Valleyview”}}(account)$$

Vertical Fragmentation of *employee_info* Relation

branch_name	*customer_name*	*tuple_id*
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

$deposit_1 = \Pi_{branch\_name,\ customer\_name,\ tuple\_id}(employee\_info)$

account_number	*balance*	*tuple_id*
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$deposit_2 = \Pi_{account\_number,\ balance,\ tuple\_id}(employee\_info)$

Advantages of Fragmentation

Horizontal:

- allows parallel processing on fragments of a relation
- allows a relation to be split so that tuples are located where they are most frequently accessed

Vertical:

- allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed
- tuple-id attribute allows efficient joining of vertical fragments
- allows parallel processing on a relation

Vertical and horizontal fragmentation can be mixed.
Fragments may be successively fragmented to an arbitrary depth.

Data Transparency

Data transparency:

Degree to which system user may remain **unaware of the details** of how and where the data items are stored in a distributed system

Consider transparency issues in relation to:

- **Fragmentation transparency**
- **Replication transparency**
- **Location transparency**

Security Consideration

- Physical database integrity
- Logical database integrity
- Element integrity
- Auditability
- Access control
- User authentication
- Availability

Security Consideration

Physical database integrity

- **immunity** to physical catastrophe, such as power failures, media failure
- physical **securing hardware**, UPS
- regular **backups**

Logical database integrity

- **reconstruction** Ability
- maintain a **log** of transactions
- replay log to restore the systems to a **stable point**

Security Consideration

Element integrity

integrity of specific database elements is their **correctness** or **accuracy**

field checks
- allow only acceptable values

access controls
- allow only authorized users to update elements

change log
- used to undo changes made in error

referential Integrity (key integrity concerns)

two phase locking process

Audit-ability

log read/write to database

Security Consideration

Access Control (similar to OS)

- logical separation by **user access privileges**
- more **complicated** than OS due to complexity of DB (granularity/inference/aggregation)

User Authentication

- may be separate from OS
- can be **rigorous**

Availability

- **concurrent** users
- **granularity** of locking
- **reliability**

SQL Security Model

SQL security model implements DAC based on

users: users of database - *user identity* checked during login process;

actions: including **SELECT, UPDATE, DELETE** and **INSERT**;

objects: tables (*base relations*), views, and columns (*attributes*) of tables and views

Users can protect objects they own

when object created, a user is designated as ‘owner’ of object

owner may grant access to others

users other than owner have to be granted *privileges* to access object

SQL Security Model

Components of privilege are

grantor, grantee, object, action, grantable

privileges managed using **GRANT** and **REVOKE** operations

the right to grant privileges can be granted

Issues with privilege management

each grant of privileges is to an individual or to “Public”

makes security administration in large organizations difficult

individual with multiple roles may have too many privileges for one of the roles

SQL Security Model

Authentication & identification mechanisms

CONNECT <user> USING<password>

DBMS may chose OS authentication or its own authentication mechanism

→ Kerberose KDC(Key Distribution Center)

→ PAM(Pluggable authentication module)

SQL Security Model

Access control through views

many security policies better expressed by granting privileges to views derived from base relations

example

CREATE VIEW AVSAL(DEPT, AVG)
AS SELECT DEPT, AVG(SALARY)
FROM EMP GROUP BY DEPT

access can be granted to this view for every dept mgr

example

CREATE VIEW MYACCOUNT AS
*SELECT * FROM Account*
WHERE Customer = current_user()

view containing account info for current user

SQL Security Model

Advantages of views

- views are flexible, and allow access control to be defined at a description level appropriate to application
- views can enforce context and data-dependent policies
- data can easily be reclassified

SQL Security Model

Disadvantages of views

- access checking may become complex
- views need to be checked for correctness (do they properly capture policy?)
- completeness and consistency not achieved automatically - views may overlap or miss parts of database
- security-relevant part of DBMS may become very large

MAC and DAC

- Mandatory Access Control (MAC) is a kind of access control that is based on asset classification e.g Top Secret, Secret, confidential etc...
- This is usually obtainable in high security organizations e.g Military...
- Discretionary Access Control (DAC) on the other hand is a type access control that is determined by the owner of an asset. e.g a user that creates a file determines who has read, write or execute permissions on the file.

SQL Security Model

- Enforce MAC using security labels
- assign security levels to all data
- label associated with a row
- assign a security clearance to each users
- label associated with the user
- DBMS enforces MAC
- access to a row based upon
- the label associated with that row and the label associated with the user accessing that row.

Security Consideration in ORACLE

Common System Security Issues

- User Authentication
- Server Authentication
- Authorization
- Secure Transmission (Encryption)
- Firewall
- Virtual Private Network (VPN)
- Demilitarized Zone (DMZ)

Security Consideration in ORACLE

User Authentication

- Authentication is the **process of verifying** that a user who logs into a network or database has permission to log in.
- Examples of authentication include the use of a **user name and password** when logging into a local-area network (LAN) and the use of **digital certificates** when sending or receiving secure e-mail over the Internet.
- An organization can use various types of authentication processes depending on the level of security desired and the type of network or database that is being protected. But in the end, the goal of authentication is to ensure that only **approved users can access the network or database and its resources.**

Security Consideration in ORACLE

Server Authentication

- When a client sends confidential data to a server, the client can verify that the server is secure and is the correct recipient of the client's confidential data.
- If you use the HTTPS communications mode, which uses HTTP 1.1 with SSL (secure sockets layer), data transmission is encrypted and server authentication is conducted over the Internet.
- Server authentication is accomplished using digital certificates. When a client browser connects to a server, the server presents its certificate.
- Servers are issued certificates from certifying authorities (CAs). CAs are companies that issue certificates to individuals or companies only after verifying the individual or company's identity.

Security Consideration in ORACLE

Authorization

- Authorization is the process of giving authenticated users access to the network or database resources they need.

Secure Transmission (Encryption)

- When information is transmitted over lines of communication, whether they be coaxial cable, telephone lines, fiber optics, or satellite, there is the risk that the communication can be intercepted by third parties.
- Often, the information can be intercepted without the sender or receiver ever knowing the data was compromised.

Security Consideration in ORACLE

Secure Transmission (Encryption)

- The methods used to encrypt data vary depending on the level of security desired and the type of network over which the data is being transmitted.
- For example, symmetric encryption can be used if network speed is paramount. Popular symmetric cryptosystems use RC-4(Rivest Cipher 4) and Data Encryption Standard (DES).
- Asymmetric encryption is highly secure, but costs in network performance. Popular asymmetric cryptosystems use Diffie-Hellman (DH) and Rivest Shamir Adlemen (RSA).

Security Consideration in ORACLE

Firewall

- A firewall is usually a combination of hardware and software that filters the types of data that can be received by your network.
- For example, a firewall can be configured to allow only HTTP traffic through to the protected network.
- A firewall also keeps your network's IP address anonymous so that it is not accessible to outside computers.

Security Consideration in ORACLE

Virtual Private Network (VPN)

- A Virtual Private Network (VPN) is an authenticated connection between two networks or between a network and a remote user where communication is considered completely private.
- Often, a VPN setup includes a firewall.
- Special "tunneling" software on both the network and the remote user's computer create a secure, encrypted connection over public lines -- even via an Internet Service Provider (ISP)

Security Consideration in ORACLE

Demilitarized Zone (DMZ)

- A Demilitarized Zone (DMZ) is an isolated environment in your network that does not contain confidential information.
- For example, you may have a network where application servers are within the demilitarized zone, but all database servers are within the protected network.
- Then, if the demilitarized zone's security is compromised, confidential data is not exposed to the intruder.

The Relational Model of Data and RDBMS Implementation Techniques