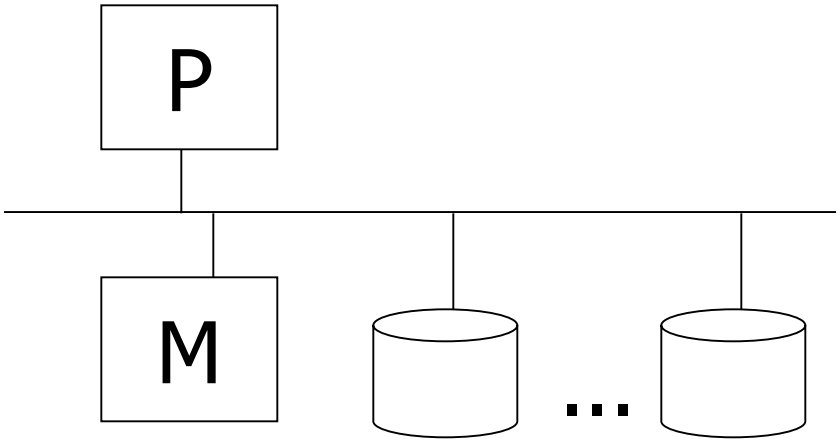
Distributed Database

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

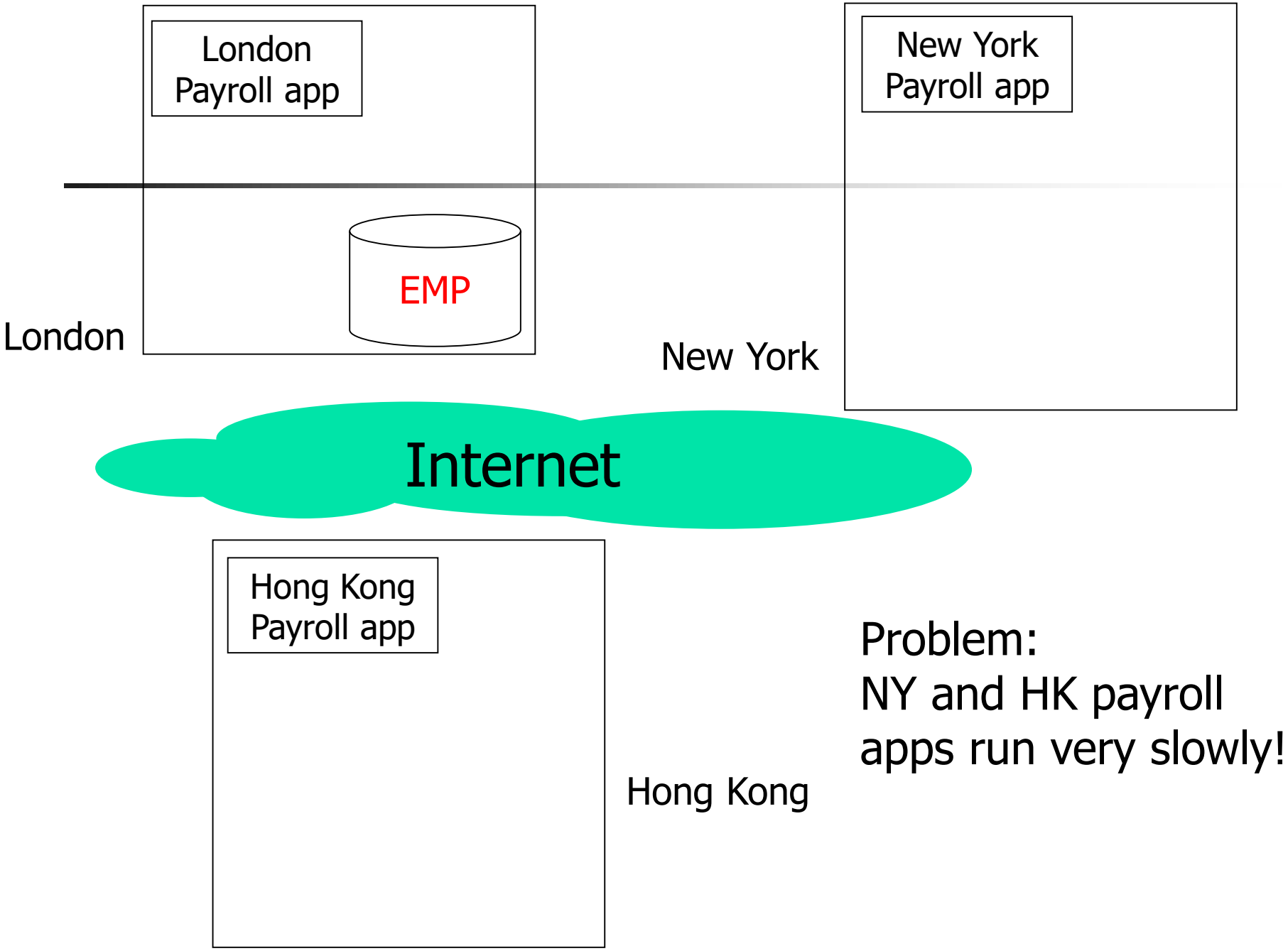
- A distributed database system consists of loosely coupled sites that share no physical component
- Database systems that run on each site are independent of each other
- Transactions may access data at one or more sites
- Multiple processors (+ memories)
- Heterogeneity and autonomy of "components"

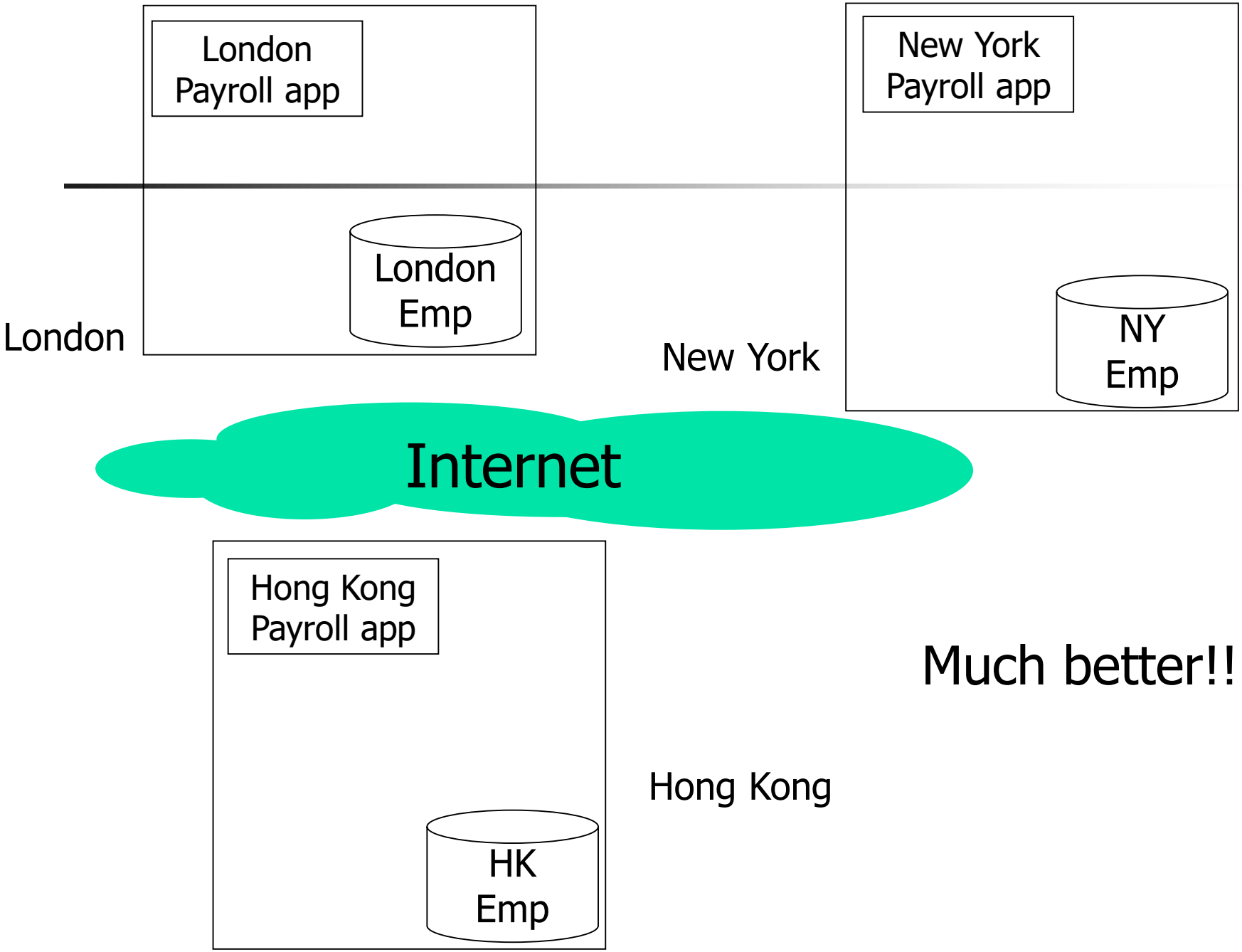
Why do we need Distributed Databases?

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

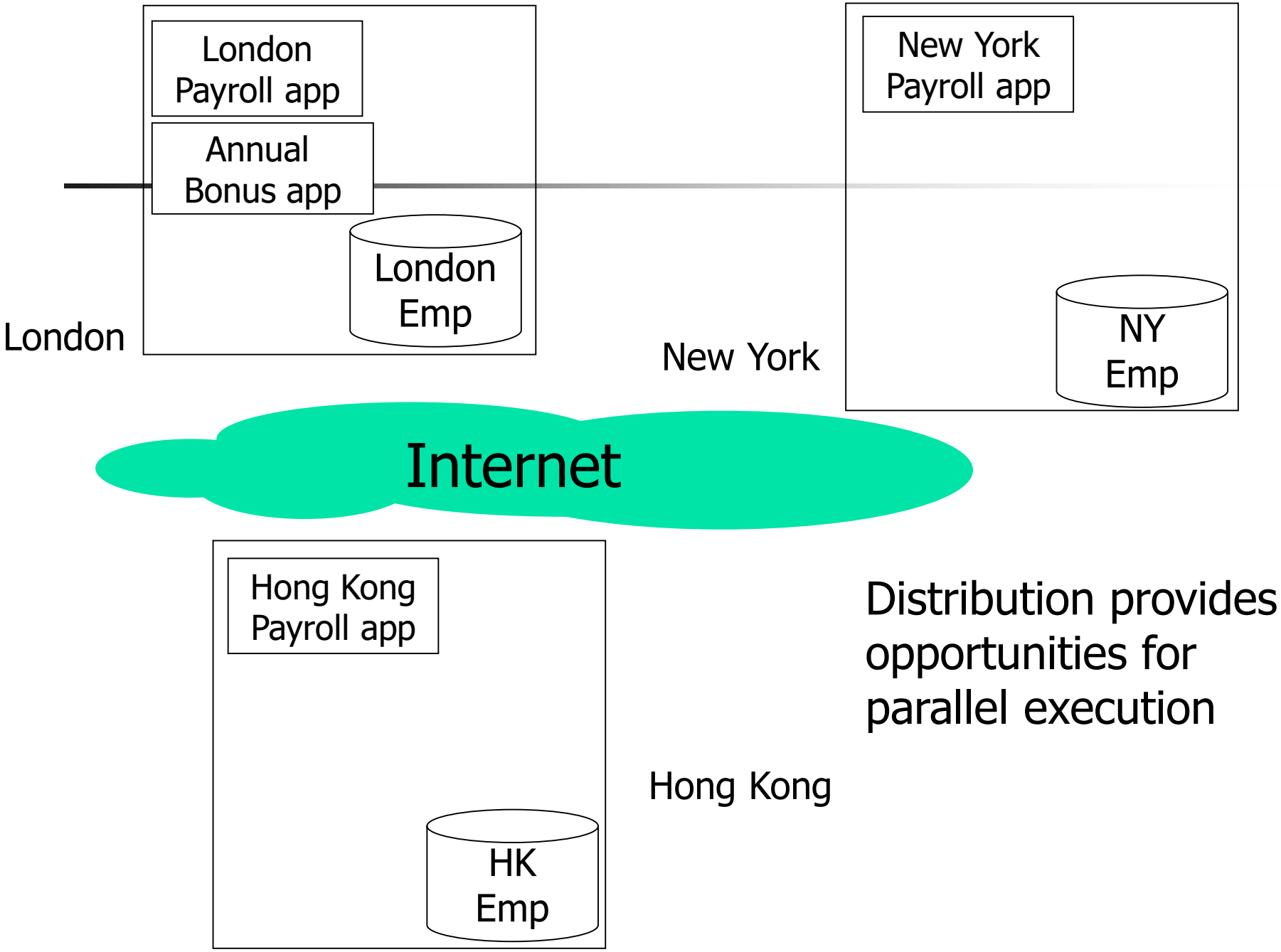
Data Access Pattern

- Mostly, employee data is managed at the office where the employee works
  - E.g., payroll, benefits, hire and fire
- Periodically, company needs consolidated access to employee data
  - E.g., company changes benefit plans and that affects all employees.
  - E.g., Annual bonus depends on global net profit.

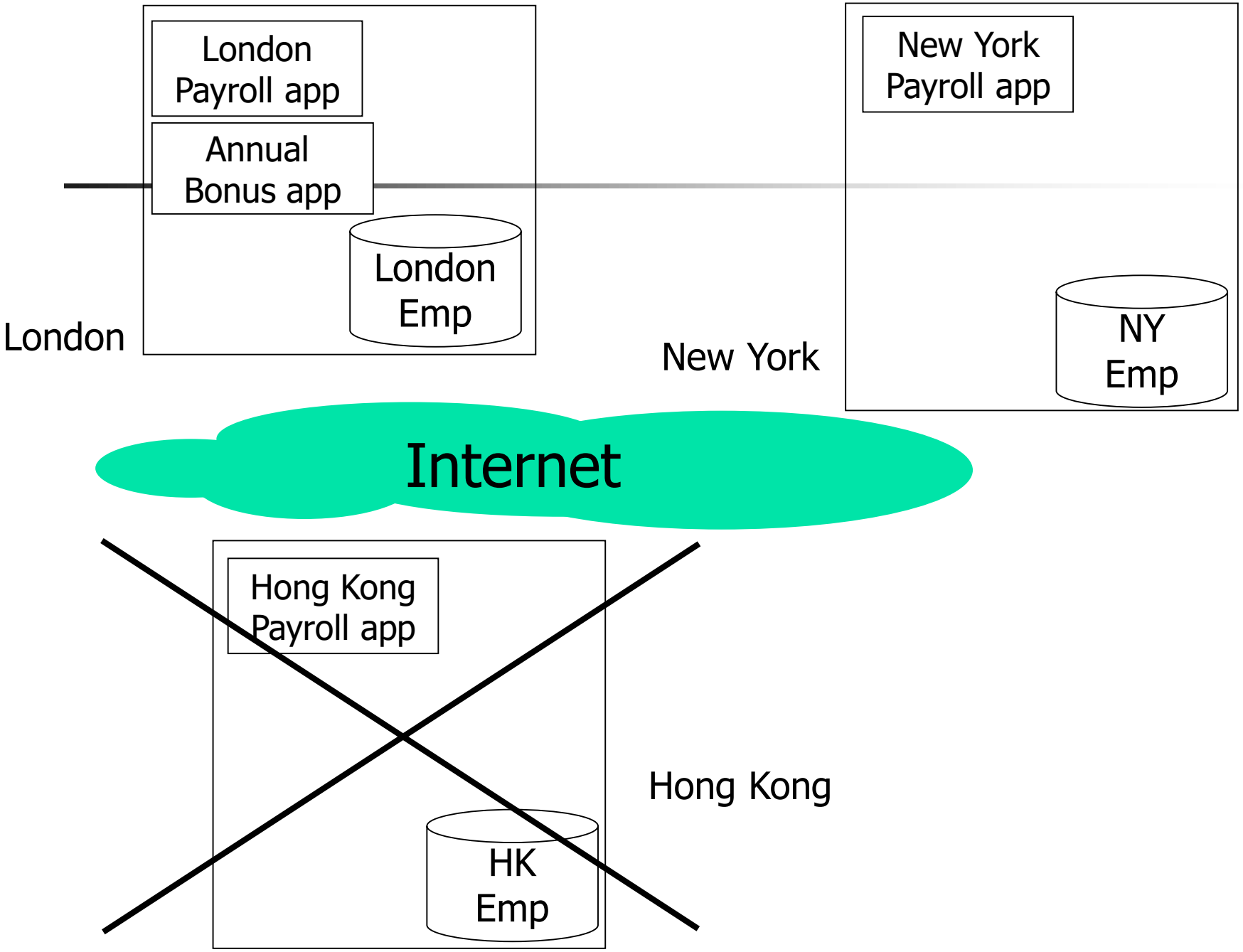

London
Payroll app
EMP
London
New York
Payroll app
New York
Internet
Hong Kong
Payroll app
Hong Kong
Problem:
NY and HK payroll
apps run very slowly!

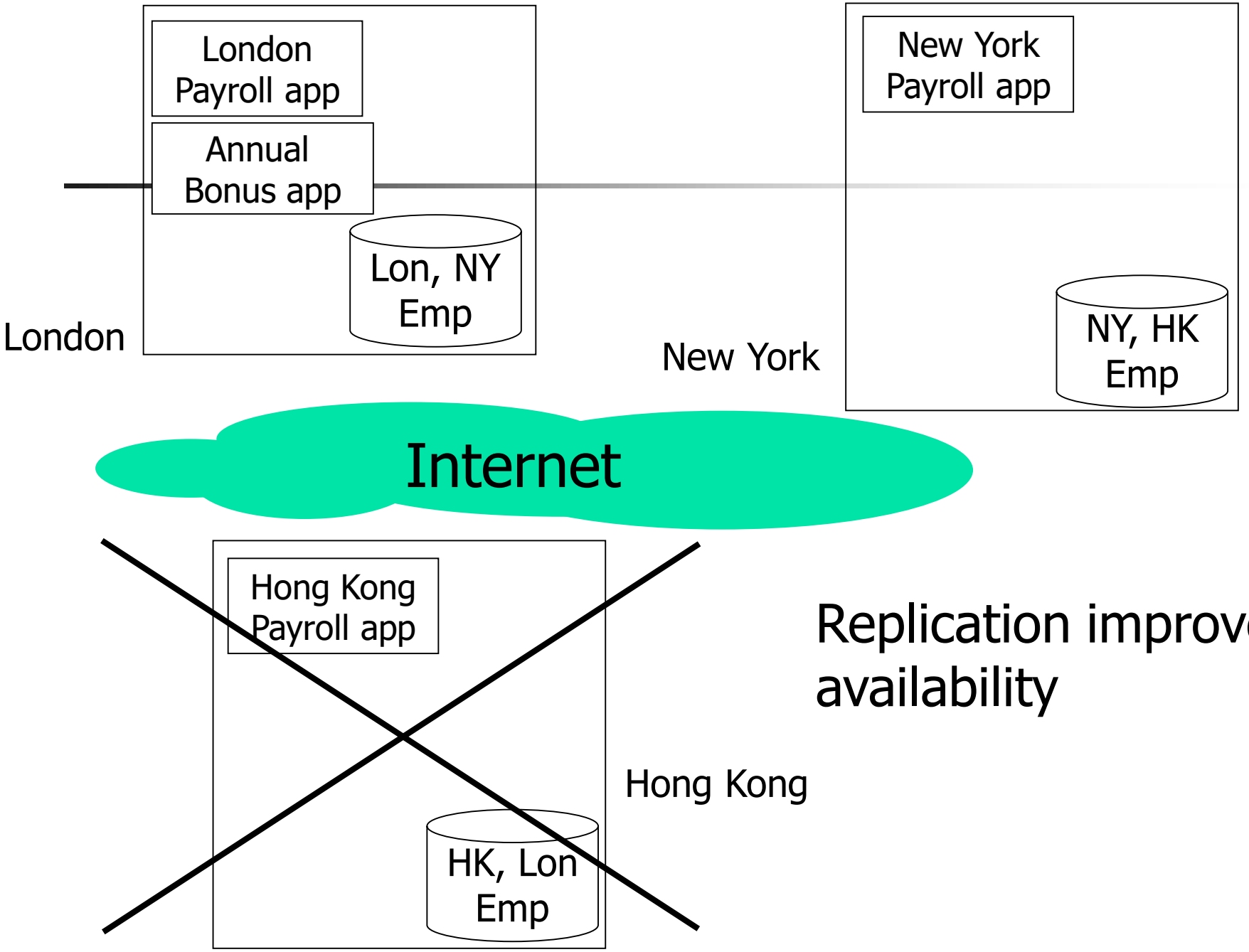

London
Payroll app
London
Emp
London
New York
Payroll app
NY
Emp
New York
Internet
Hong Kong
Payroll app
HK
Emp
Hong Kong

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



Replication improves availability

Distributed Database Features

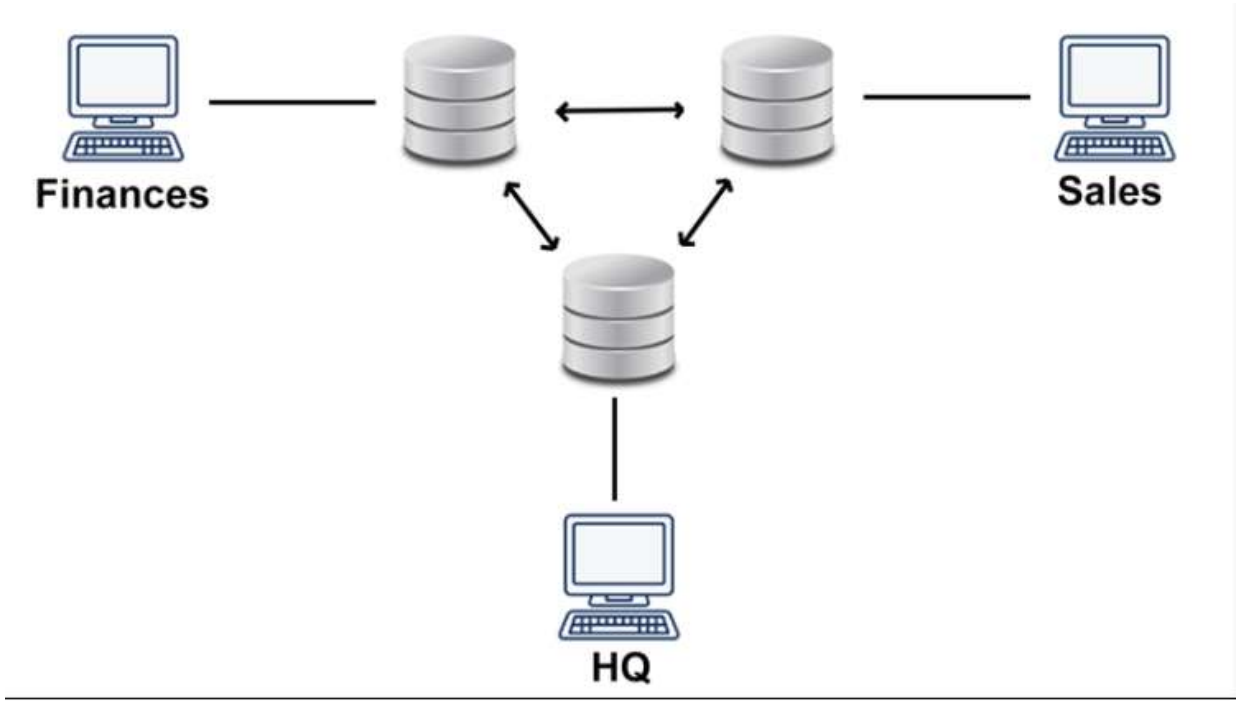
Some general features of distributed databases are:

- **Location independency** - Data is physically stored at multiple sites and managed by an independent DDBMS.
- **Distributed query processing** - Distributed databases answer queries in a distributed environment that manages data at multiple sites. High-level queries are transformed into a query execution plan for simpler management.
- **Distributed transaction management** - Provides a consistent distributed database through commit protocols, distributed concurrency control techniques, and distributed recovery methods in case of many transactions and failures.
- **Seamless integration** - Databases in a collection usually represent a single logical database, and they are interconnected.
- **Network linking** - All databases in a collection are linked by a network and communicate with each other.
- **Transaction processing** - Distributed databases incorporate transaction processing, which is a program including a collection of one or more database operations. Transaction processing is an atomic process that is either entirely executed or not at all.

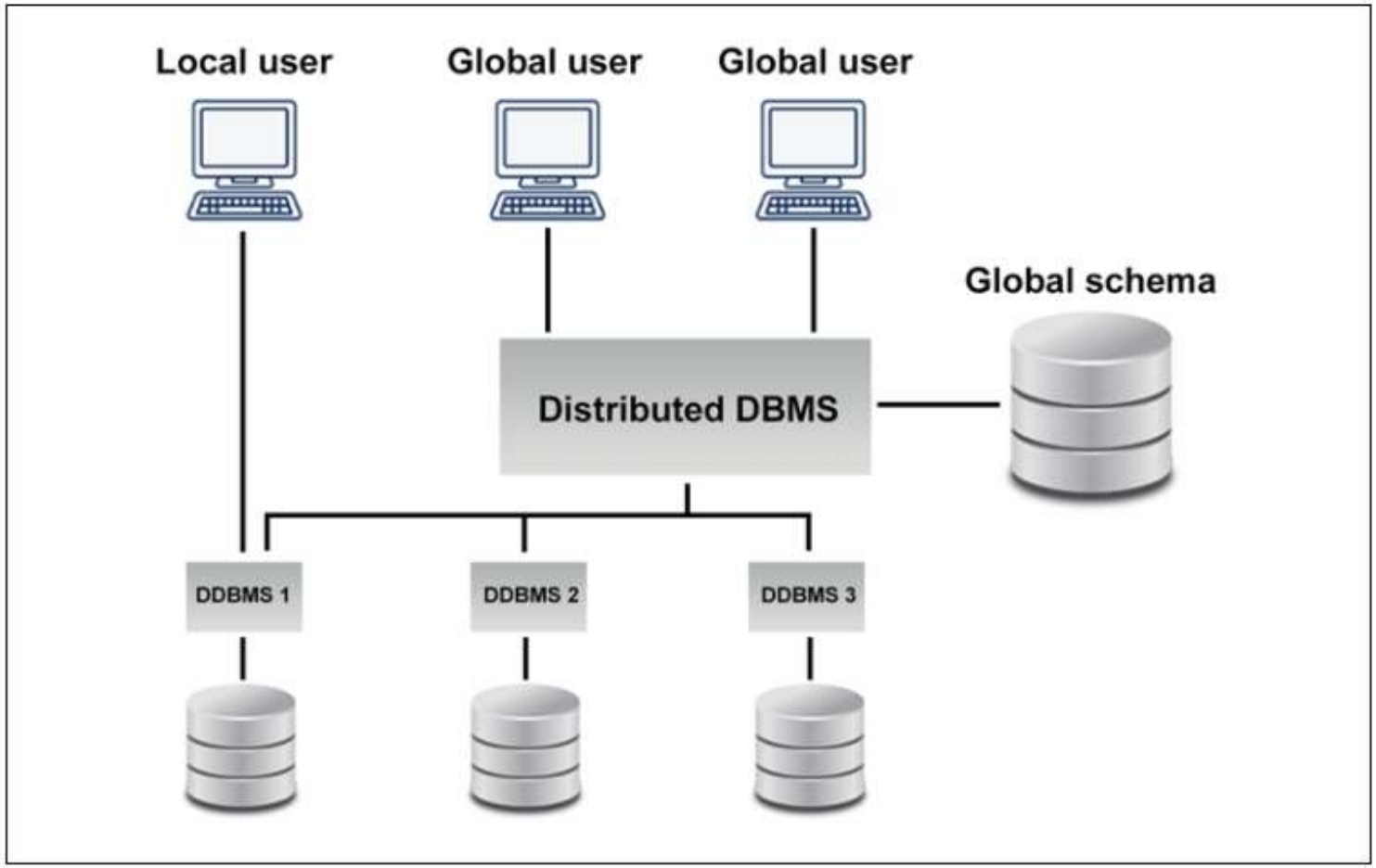
Distributed Database Types

- **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

homogeneous distributed database



heterogeneous distributed database



Distributed Database Advantages and Disadvantages

Below are some key advantages and disadvantages of distributed databases:

Advantages	Disadvantages
Modular development	Costly software
Reliability	Large overhead
Lower communication costs	Data integrity
Better response	Improper data distribution

Distributed Database Challenges

- Distributed Database Design
  - Deciding what data goes where
  - Depends on data access patterns of major applications
  - Two subproblems:
    - 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

$$account_1 = \sigma_{branch\_name="Hillside"}(account)$$

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

$$account_2 = \sigma_{branch\_name="Valleyview"}(account)$$

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

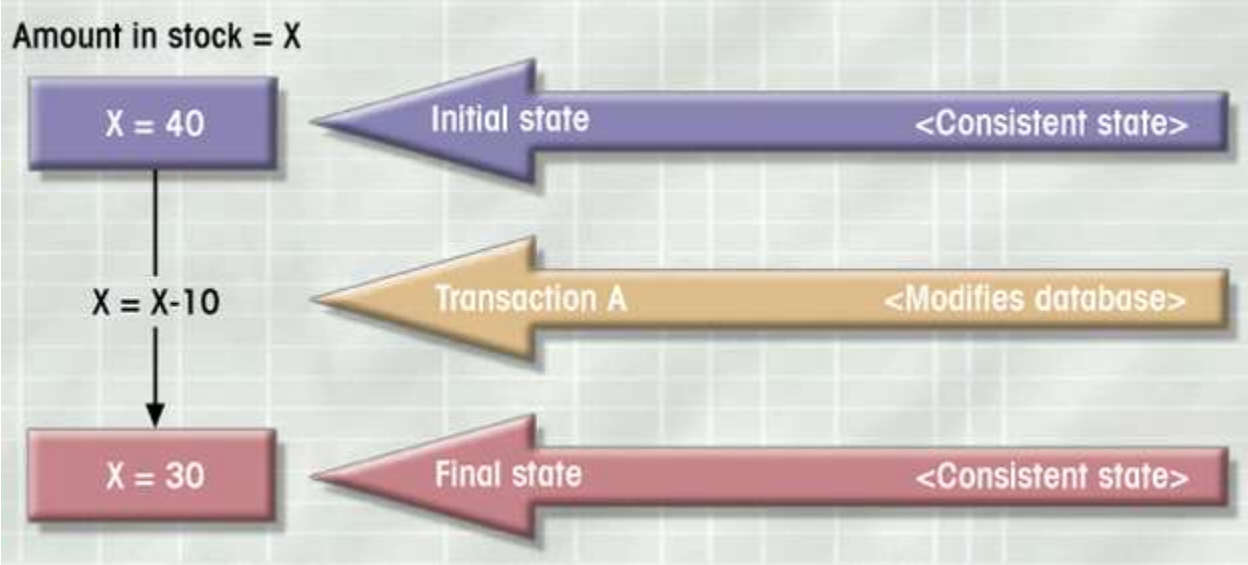
- Horizontal:
  - allows parallel processing on fragments of a relation
  - allows a relation to be split so that tuples are located where they are most frequently accessed
- Vertical:
  - allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed
  - tuple-id 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

What is a Transaction?

- A set of steps completed by a DBMS to accomplish a single user task.
- Must be either entirely completed or aborted
- No intermediate states are acceptable



Distributed Transactions

- Transaction may access data at several sites.
- 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 transactions that originate at the site.
  - Distributing subtransactions at appropriate sites for execution.
  - Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

Distributed Query Processing

- For centralized systems, the primary criterion for measuring the cost of a particular strategy is the number of disk accesses.
- In a distributed system, other issues must be taken into account:
  - The cost of a data transmission over the network.
  - The potential gain in performance from having several sites process parts of the query in parallel.

Query Processing

- Input: Declarative Query
  - SQL, OQL, XQuery, ...
- Step 1: Translate Query into Algebra
  - Tree of operators
- Step 2: Optimize Query (physical and logical)
  - Tree of operators
  - (Compilation)
- Step 3: Interpretation
  - Query result

Conclusion- Advantages of DDBMSs

- Reflects organizational structure
- Improved shareability and local autonomy
- Improved availability
- Improved reliability
- Improved performance
- Economics
- Modular growth

Conclusion- Disadvantages of DDBMSs

- ☹ Architectural complexity
- ☹ Cost
- ☹ Security
- ☹ Integrity control more difficult
- ☹ Lack of standards
- ☹ Lack of experience
- ☹ Database design more complex