Databases

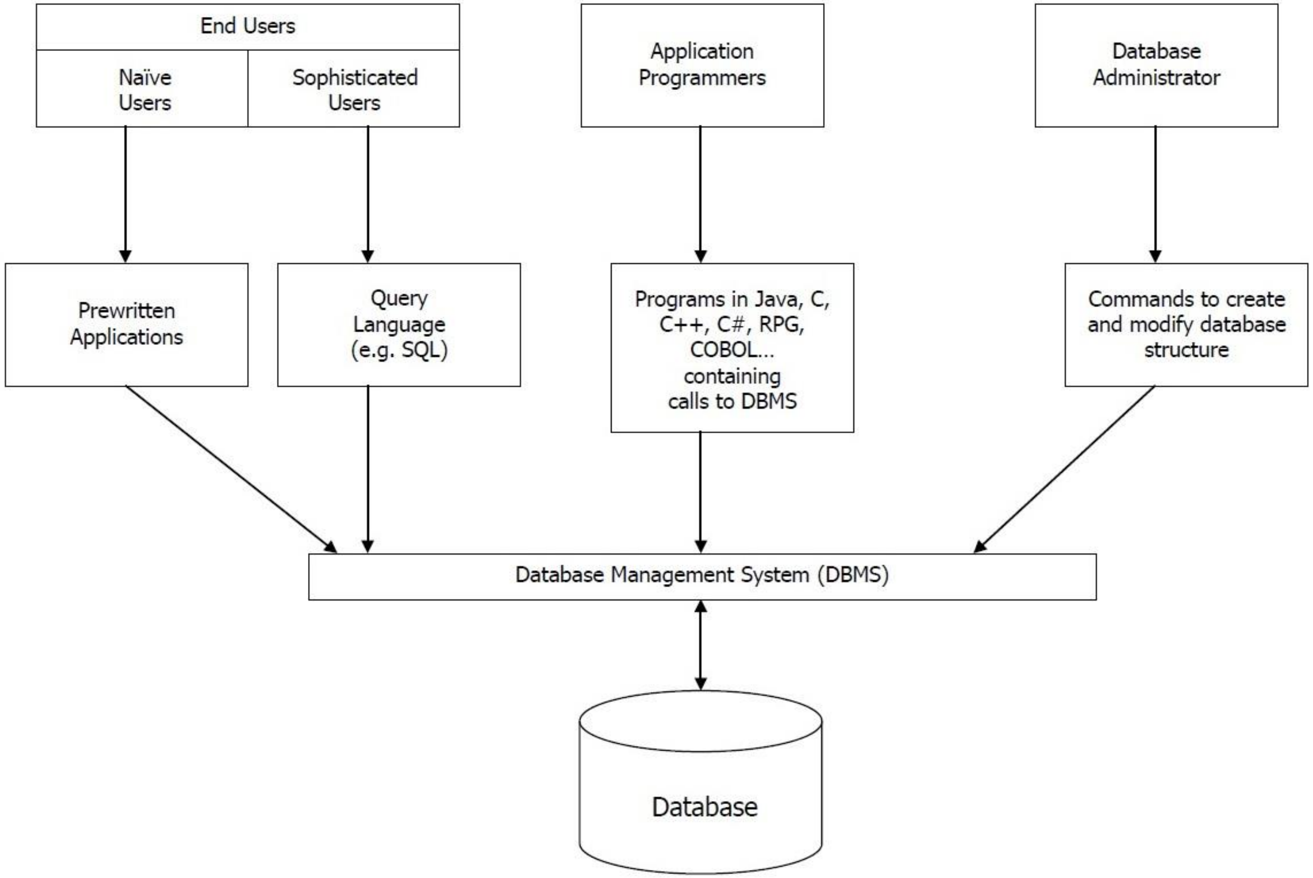
Distributed Architectures and DBMS

Dr Konrad Dabrowski
konrad.dabrowski@durham.ac.uk

Online Office Hour:
Mondays 13:30–14:30
See Duo for the Zoom link

Multi-user DBMS Architectures

- So far we have seen this model of interaction between end-users and the database:
  - one central DBMS
  - users interact with DBMS using an application program

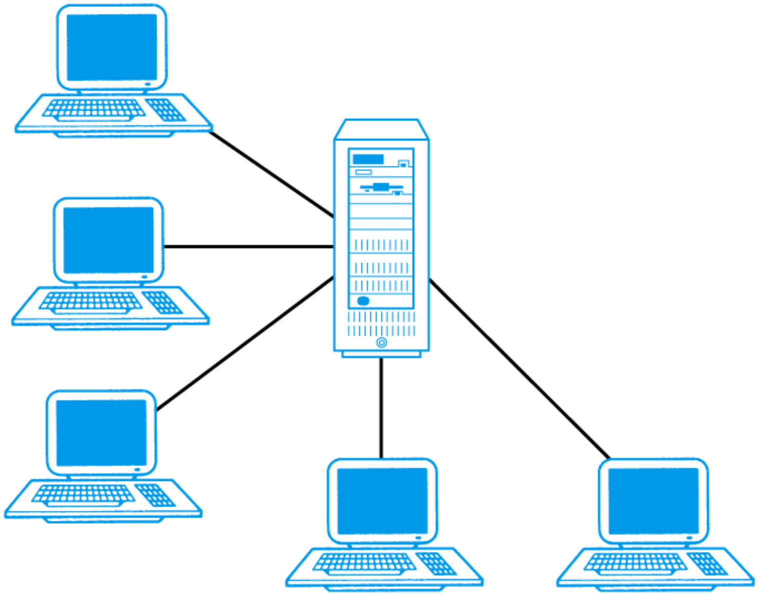


Multi-user DBMS Architectures

- Several issues still need clarification:
  - is the DBMS on the end-user's computer?
  - how are the users connected to the DBMS?
    (e.g. physically, through LAN etc.)
  - do we have one or more DBMS?
  - which computations are performed where?
    (e.g. on the user's computer, on some server etc.)
  - is the database stored in one or many places?
- The answers are not unique:
  - different company needs require different approaches

Multi-user DBMS Architectures

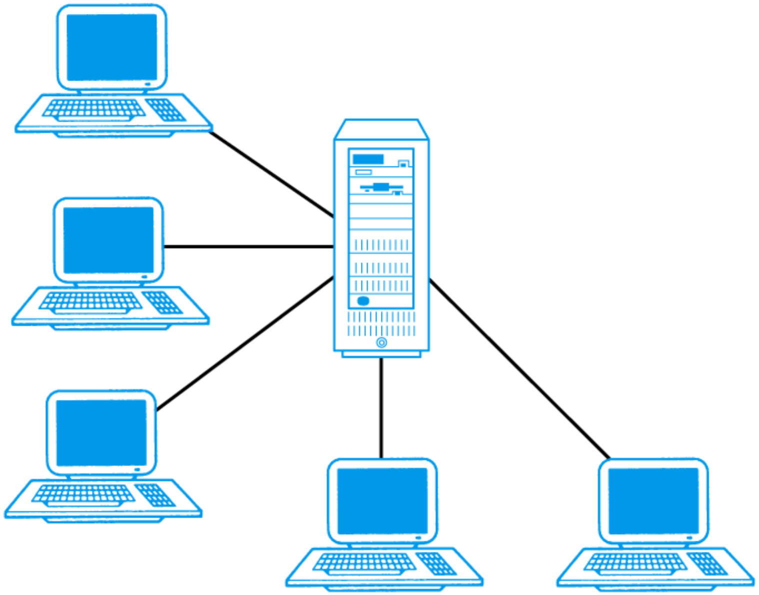
- **Teleprocessing Architecture:**
  - the traditional (and most basic) architecture
  - one computer with a single Central Processing Unit (CPU)
  - many (end-user) terminals,
    all cabled to the central computer
  - a terminal sends messages
    to the central computer
    (through the user's application program)
  - all data processing
    in the central computer:
    - receive messages from terminal
    - process the queries
    - send the results back to the terminal



Multi-user DBMS Architectures

- **Teleprocessing Architecture:**
  - tremendous burden on the central computer

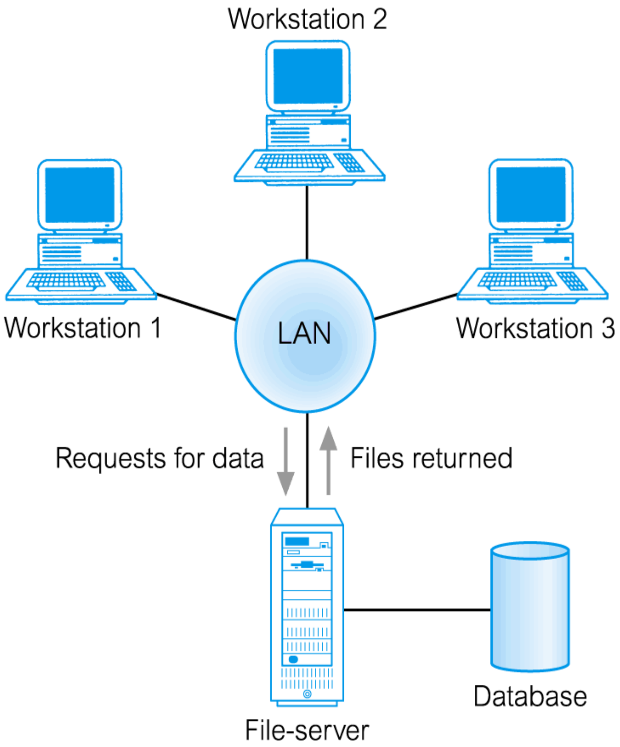
  $\Longrightarrow$ decreased performance
- Nowadays, the trend is towards downsizing:
  - replace expensive mainframe computers with cost-effective networks of personal computers
  - achieve the same / better results



- The next two architectures:
  - file-server architecture
  - client-server architecture

File-Server architecture

- Processing is distributed around a computer network
  - typically through a Local Area Network (LAN)
  - one central file-server
    (computer containing the database)
  - every workstation has its own DBMS
    and its own user application
  - workstations request files they need
    from the file server
  - file server acts like a "shared hard disk"
    (it has no DBMS!)



- For example, the user request is:
  - "find the names of staff who work in the branch at 163 Main St."
  - the user application creates an SQL query for this

File-Server architecture

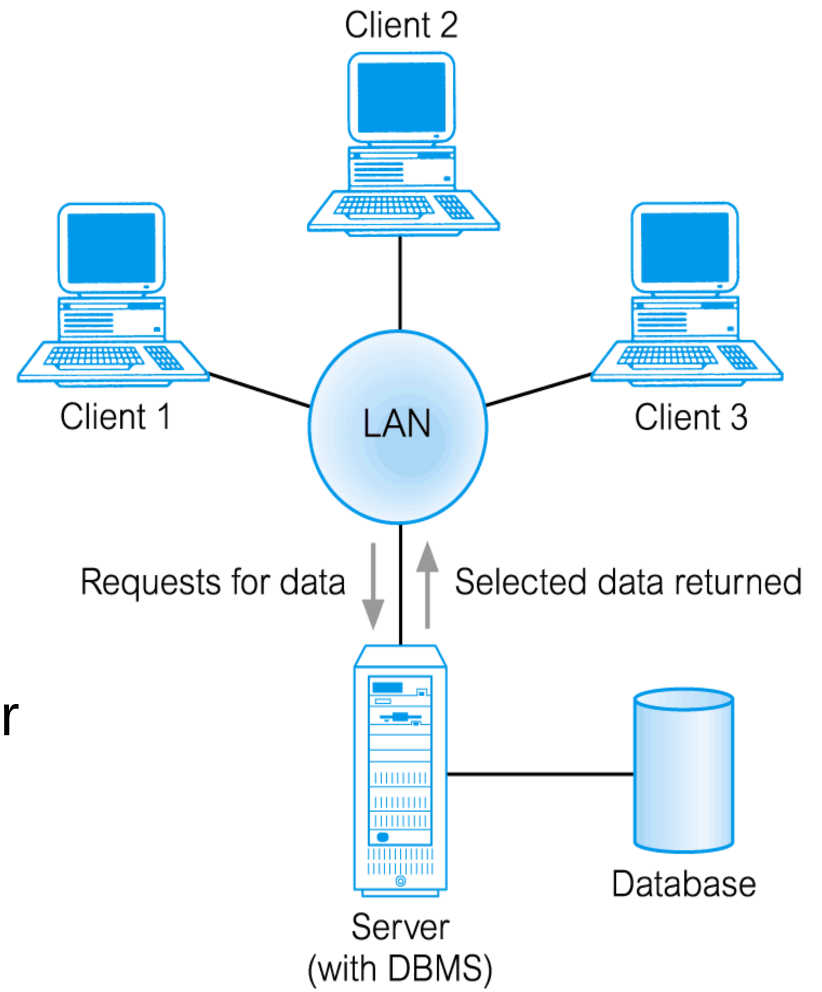
- The SQL query:

```
SELECT  fName, lName
FROM    Branch b, Staff s
WHERE   b.branchNo = s.branchNo AND b.street = '163 Main St.'
```

- The file-server has no knowledge of SQL (has no DBMS!)
  - the user's DBMS has to request the whole tables Branch, Staff
- Therefore:
  - very large amount of network traffic (the tables may be huge)
  - a full copy of the DBMS is required on each workstation
  - concurrency / recovery / integrity control is more difficult, since multiple DBMSs access the same files simultaneously
- The solution to these problems:
  - client-server architectures

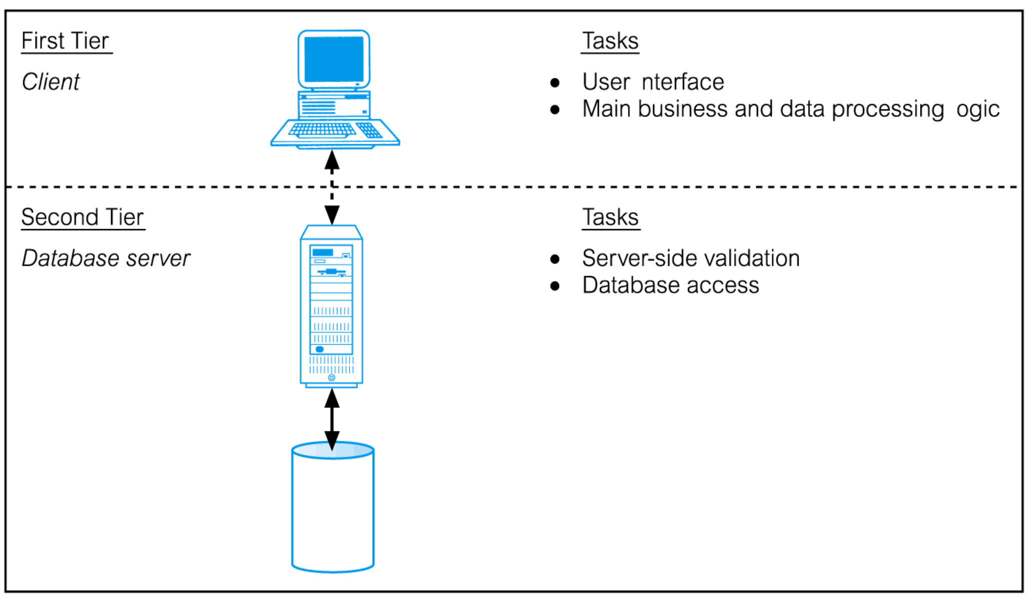
Client-Server architecture

- Client: requires some resource
- Server: provides the resource
- Client / Server are not always in the same machine / place
- Two-tier architecture:
  - Tier 1 (client): responsible for the presentation of data to the user
  - Tier 2 (server): responsible for supplying data services to the user
- Typical procedure:
  - user gives a request to the client
  - client generates SQL query and sends it to the server
  - server accepts, processes the query and sends the result to the client
  - client formats the result for the user



Client-Server architecture

- Many advantages:
  - increased performance: many client CPUs work in parallel
  - reduced hardware costs: only the server needs increased storage and computational power
  - reduced communication costs: less data traffic (no unnecessary data transmitted)
- Database is still centralized:
  - not a distributed database!



Three-Tier Client-Server Architecture

- In modern systems: 100s / 1000s of users

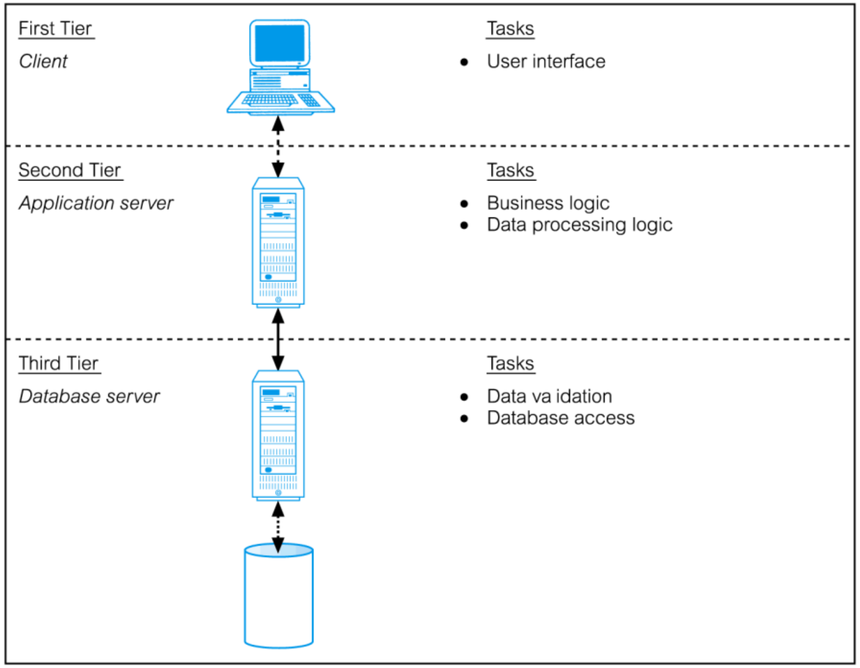
  $\Longrightarrow$ need for increased enterprise scalability

- Main problem of the client that prevent scalability:
  - a "fat client" (many users) requires extensive resources on disk space / RAM / CPU power

- A new variation of client-server architecture (1995):
  - three layers, potentially running on different platforms
  - First tier: user-interface layer (on end-user's computer)
  - Second tier: application server (connects to many users)
  - Third tier: database server
    (contains DBMS, communicates with the application server)
  - "thin clients" $\Longrightarrow$ increased performance of user's computer

Three-Tier Client-Server Architecture

- Best example for a client: internet browser (fast & light)
- Advantages:
  - smaller hardware cost for the “thin clients”
  - easier application maintenance (centralized in one tier)
  - easier to modify/replace one tier without affecting the others
  - easier load balancing between the different tiers
  - maps naturally to Web applications



- It can be extended:
  - separation of tasks into *n* intermediate tiers

  $\Longrightarrow$ increased flexibility / scalability

Distributed DBMS

- So far we have seen centralized database systems:
  - single database, located at one site
  - controlled by one DBMS
- We can improve database performance:
  - using networks of computers (decentralized approach)
  - it mirrors the organizational structure:
    - logically distributed into divisions, departments, projects…
    - physically distributed into offices, units, factories…
- Main targets:
  - make all data accessible to all units (avoid “islands of information”)
  - store the data proximate to the location where it is most frequently used

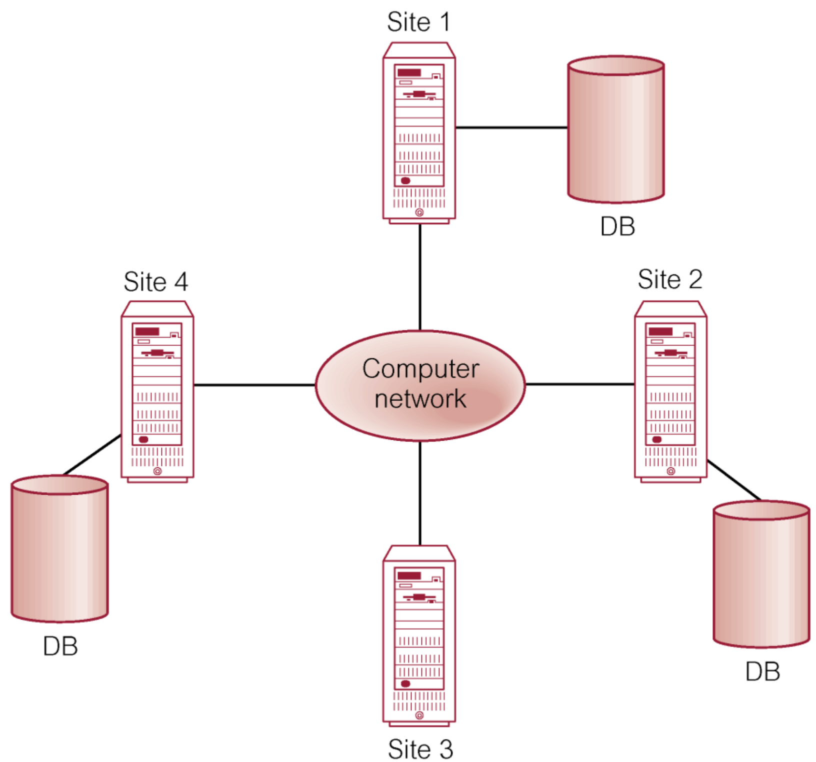
$\Longrightarrow$ full functionality and efficiency

Distributed DBMS

- Distributed database:
  - a logically interrelated collection of shared data, physically distributed over a network
- Distributed DBMS (DDBMS):
  - the software system that can manage the distributed database
  - it makes the distribution transparent (invisible) to users
- In a DDBMS:
  - a single logical database, which is split into fragments
  - each fragment is stored on one (or more) computers, under the control of a separate DBMS
  - all these computers are connected by a communications network
  - sites have local autonomy: independent processing of local data (via local applications)
  - sites have access to global applications (to process data fragments stored on other computers)

Distributed DBMS

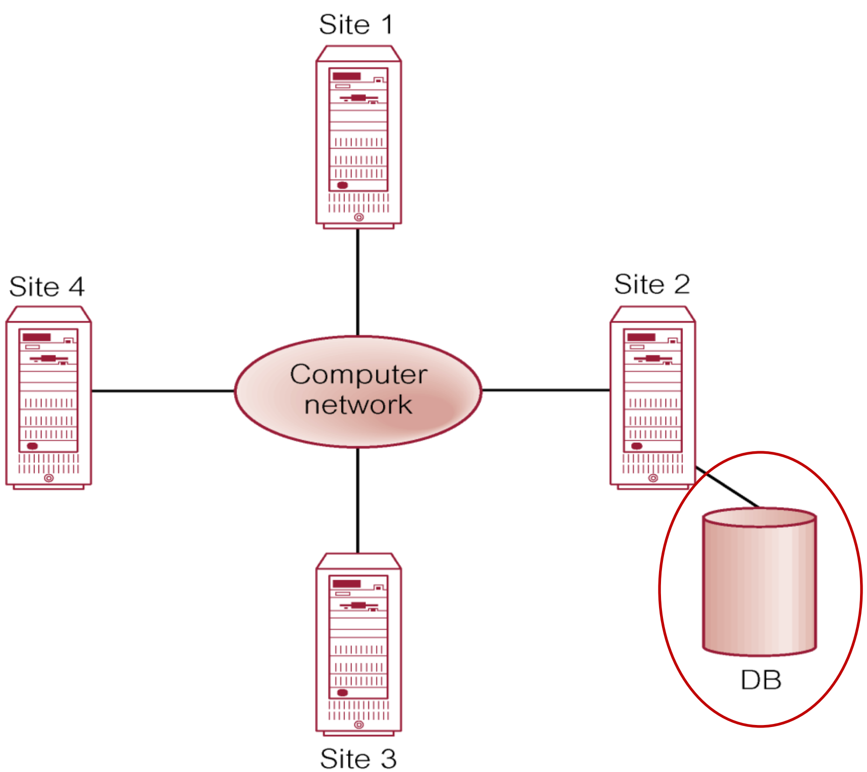
- Not all sites have local applications / local data
- All sites have access to global applications
- Data fragments may be replicated in more sites
  - data consistency must be considered
    (same values for all replications)



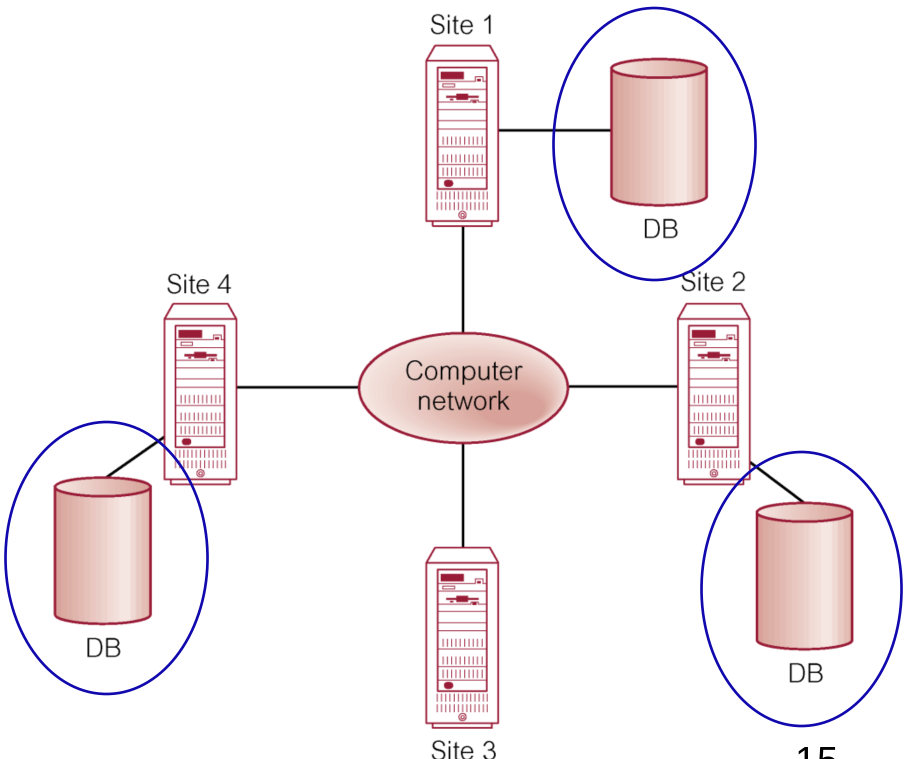
Distributed Processing vs. Distributed DBMS

- Distributed processing:
  - a centralized database that is accessed over a computer network
  - for example: the client-server architecture
  - not the same as distributed DBMS!

distributed processing:



distributed DBMS:



Design of a distributed DBMS

- In addition to ER modelling, we also have to consider:
  - Fragmentation
    - how to break a relation into fragments
    - fragments can be horizontal / vertical / mixed
  - Allocation
    - how fragments are allocated at the several sites
    - aim is to reach an "optimal" distribution (efficient, reliable…)
  - Replication
    - which fragments are stored at multiple sites (& at which sites)
- Choices for Fragmentation and Allocation:
  - based on how the database is to be used
  - quantitative and qualitative information needed

Design of a distributed DBMS

- Quantitative information: (mainly for fragmentation)
  - the frequency with which specific transactions are run
  - the (usual) sites from which transactions are run
  - desired performance criteria for the transactions
- Qualitative information: (mainly for allocation)
  - the relations / attributes / tuples being accessed
  - the type of access (read / write)
- Strategic objectives for the choices about the fragments:
  - locality of reference
    - data to be stored close to where it is used
    - if a fragment is used at several sites → replication is useful
  - reliability and availability
    - improved by replication
    - if one site fails, there are other fragment copies available

Design of a distributed DBMS

- Further strategic objectives:
  - acceptable performance
    - bad allocation results in "bottleneck" effects
      (a site receives too many requests ⇒ bad performance)
    - also: bad allocation causes underutilized resources
  - cost of storage capacities
    - cheap mass storage to be used at sites, whenever possible
    - this must be balanced against locality of reference
  - minimal communication costs
    - min. retrieval costs when max. locality of reference,
      or when each site has its own copy of data
    - but when replicated data are updated:
      → all copies of this data must be updated
      → increased network traffic / communication costs

Design of a distributed DBMS

- Four alternative strategies for the placement of data:
  - centralized
    - single database and DBMS
    - stored at one site with users distributed across the network
    - not distributed!
  - partitioned
    - database partitioned into disjoint fragments
    - each data item assigned to exactly one site (no replication!)
  - complete replication
    - complete copy of the database at each site
  - selective replication
    - combination of partitioning, replication, and centralization

Design of a distributed DBMS

- Four alternative strategies for the placement of data:

Balance of the strategic objectives

	Locality of reference	Reliability and availability	Performance	Storage costs	Communication costs
Centralized	Lowest	Lowest	Unsatisfactory	Lowest	Highest
Fragmented	High[a]	Low for item; high for system	Satisfactory[a]	Lowest	Low[a]
Complete replication	Highest	Highest	Best for read	Highest	High for update; low for read
Selective replication	High[a]	Low for item; high for system	Satisfactory[a]	Average	Low[a]

[a] Indicates subject to good design.

Design of a distributed DBMS

Three correctness rules for the partitioned placement (i.e. with no replication):

1. Completeness
   - if relation R is decomposed into fragments $R_1, R_2, \ldots, Rn$, each data item that can be found in R must appear in at least one fragment R_i
2. Reconstruction
   - it must be possible to define a rel. algebra expression that can reconstruct R from its fragments
3. Disjointness
   - if a data item appears in fragment R_i, it should not appear in any other fragment
   - exception for vertical fragmentation: primary key attributes must be repeated for the reconstruction !

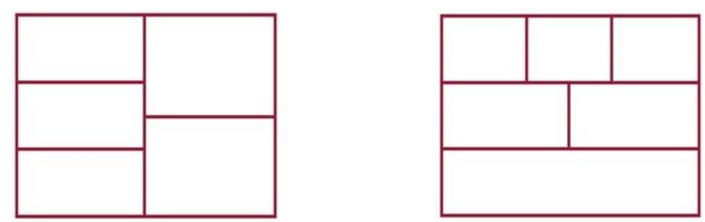
Design of a distributed DBMS

Three main types of fragmentation:

1. horizontal
   – a subset of the tuples of the relation

2. vertical
   – a subset of the attributes of the relation

3. mixed
   – a vertical fragment that is then horizontally fragmented
   – or a horizontal fragment that is then vertically fragmented



Horizontal fragmentation

- Assume there exist two property types: ‘Flat’ and ‘House’
- We have a relation R with all properties for rent
- The horizontal fragmentation of R (by property-type) is:
  - $P_1 = \sigma_{\text{type='House'}}(\text{PropertyForRent})$
  - $P_2 = \sigma_{\text{type='Flat'}}(\text{PropertyForRent})$

Relational Algebra operator “σ”: “select” specific rows of a table

- This fragmentation may be useful:
  - e.g. if we have separate applications dealing with flats / houses
- and it is correct:
  - completeness: each tuple is either in P_1 or in P_2
  - reconstruction: R can be reconstructed from the fragments P_1, P_2

$$R = P_1 \cup P_2$$

  - disjointness: there is no property that is both ‘flat’ and ‘house’

Vertical fragmentation

- For every staff member in a company:
  - the payroll department requires: staffNo, position, sex, salary
  - the HR department requires: staffNo, fName, DOB, branchNo
- We have a relation *Staff* with all staff members
- For this example, the vertical fragmentation of *Staff* is:
  - $S_1 = \Pi_{\text{staffNo, position, sex, salary}}(\text{Staff})$
  - $S_2 = \Pi_{\text{staffNo, Name, DOB, branchNo}}(\text{Staff})$

Relational Algebra operator "∏" : "project" specific columns of a table

- Both fragments include the primary key staffNo
  - to allow reconstruction of *Staff* from S_1 and S_2
- This fragmentation is useful:
  - fragments are stored at the departments that are needed
  - performance for every department is improved (as the fragment is smaller than the original relation *Staff*)

Vertical fragmentation

$S_1 = \Pi_{\text{staffNo, position, sex, salary}}(\text{Staff})$

$S_2 = \Pi_{\text{staffNo, Name, DOB, branchNo}}(\text{Staff})$

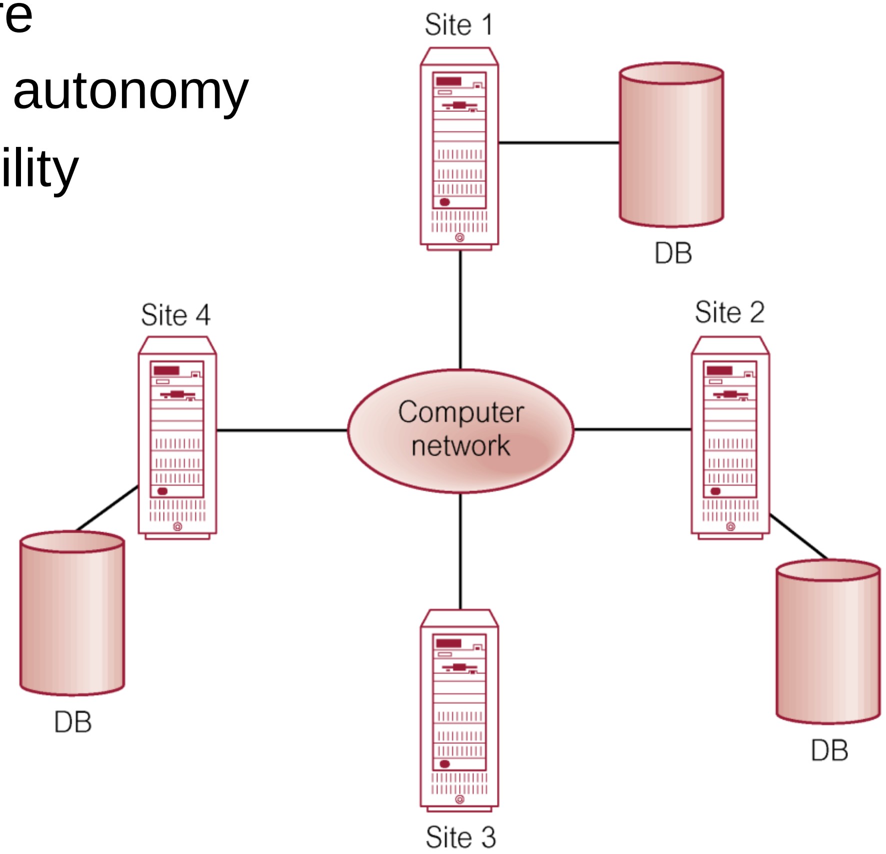
- This fragmentation is correct:
  - completeness:
    - the primary key staffNo belongs to both S_1 and S_2
    - each other attribute is either in S_1 or in S_2
  - reconstruction: R can be reconstructed from the fragments S_1, S_2 using the Natural join operation:

$$Staff = S_1 \bowtie S_2$$

  - disjointness: the fragments are disjoint except the primary key (which is necessary for the reconstruction)

(Dis-)advantages of distributed DBMS

- Advantages:
  - Reflects organizational structure
  - Improves shareability and local autonomy
  - Improved availability and reliability
  - Improved performance
  - Smaller hardware cost
  - Scalability



- Disadvantages:
  - Complexity
  - Higher maintenance cost
  - Security
  - Integrity control more difficult
  - Design more complex
  - Lack of experience in the industry