Database Systems for Big Data

Iztok Savnik, FAMNIT

Course literature

- Textbook
  - Tamer Özsu, Patrick Valduriez, Principles of Distributed Database Systems, 4th Edition, Springer, ISBN 978-1-4419-8833-1, 2020.
- Transparences
  - Tamer Özsu, Patrick Valduriez: based on the textbook
  - Presentations of NoSQL and NewSQL systems
- Research papers
  - In the 2nd part of the course, each topic will include a list of papers.

Grading

- Exam (written) = 40%
  - 120 min, 4 exercises
  - >50%!
- Seminar = 40%
  - Study of a research paper (new approach, survey, active research)
  - Study of a novel DBMS (test app (distrbd), report, presentation)
  - >50%!
- Quizzes = 20%
  - 2-3 questions about the topics from the previous lecture
  - 15 min - At the beginning of each lecture
  - Grade = The average of the best 80% grades of quizzes

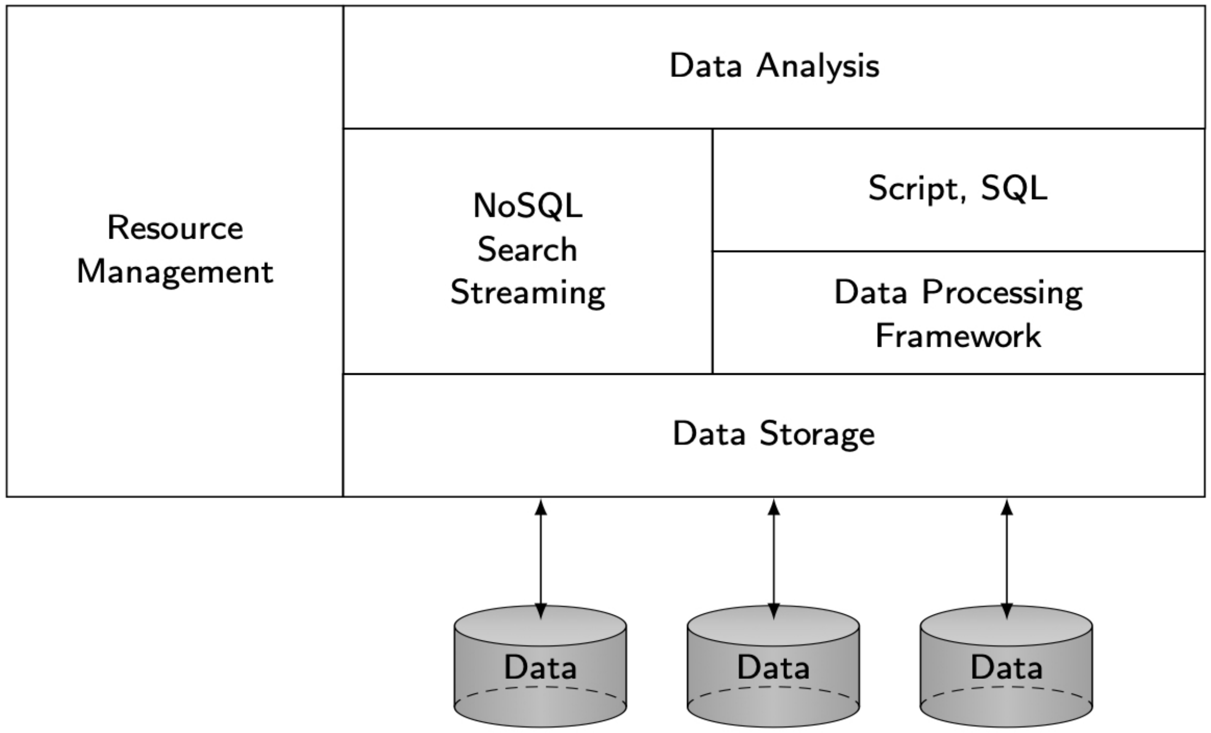
Outline

- Introduction
  - Big data
  - What is a distributed DBMS?s
  - History
  - DDBMS promises
  - DDBMS issues
  - DDBMS architecture
  - New database systems

Four Vs

- Volume
  - Increasing data size: petabytes (10^{15}) to zettabytes (10^{21})
- Variety
  - Multimodal data: structured, images, text, audio, video
  - 90% of currently generated data unstructured
- Velocity
  - Streaming data at high speed
  - Real-time processing
- Veracity
  - Data quality

Big Data Software Stack



Big data database systems

- Distributed database systems
  - One server can not store everything
- Relational distributed DBMSs
  - IBM, Oracle, Sybase
  - Oldest lineage in database area
- NoSQL database systems
  - Key/Value store
  - Columnar DBMS
  - Document store
  - Graph DBMS
- NewSQL systems
  - Distributed Relational DBMSs
  - Google F1, SAP Hana, VoltDB

Big Data Analytics

- Map-Reduce/Spark systems
  - Distributed file systems (GFS, Hadoop)
  - A query is a graphs of operators
  - Tree-structured computation
- Stream query processing
  - Data streams
  - Stream QLs
  - Persistent queries
- Data-flow systems
  - Programming environments
  - Based on data-flow
  - Directed Acyclic graphs (DAGs)

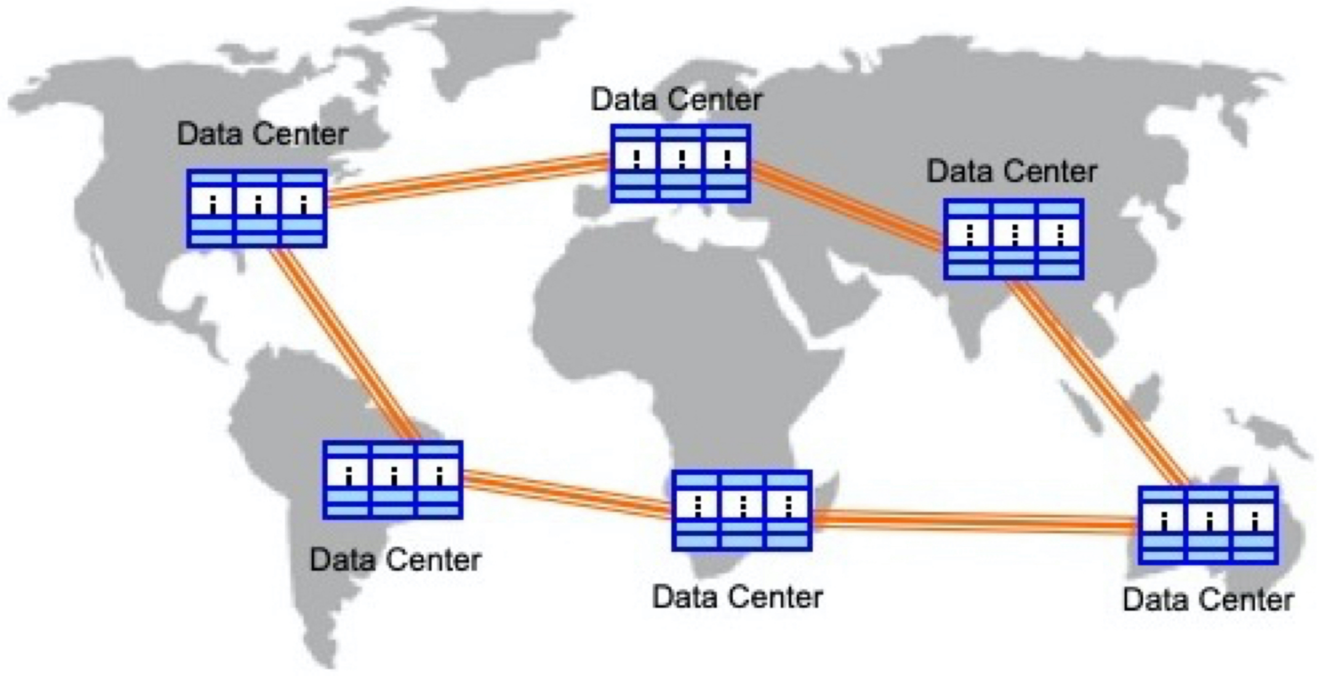
Outline

- Introduction
  - Big data
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - DDBMS issues
  - Distributed DBMS architecture
  - New database systems

Distributed Computing

- A number of autonomous processing elements (not necessarily homogeneous) that are interconnected by a computer network and that cooperate in performing their assigned tasks.
- What is being distributed?
  - Processing logic
  - Function
  - Data
  - Control

Current Distribution – Geographically Distributed Data Centers

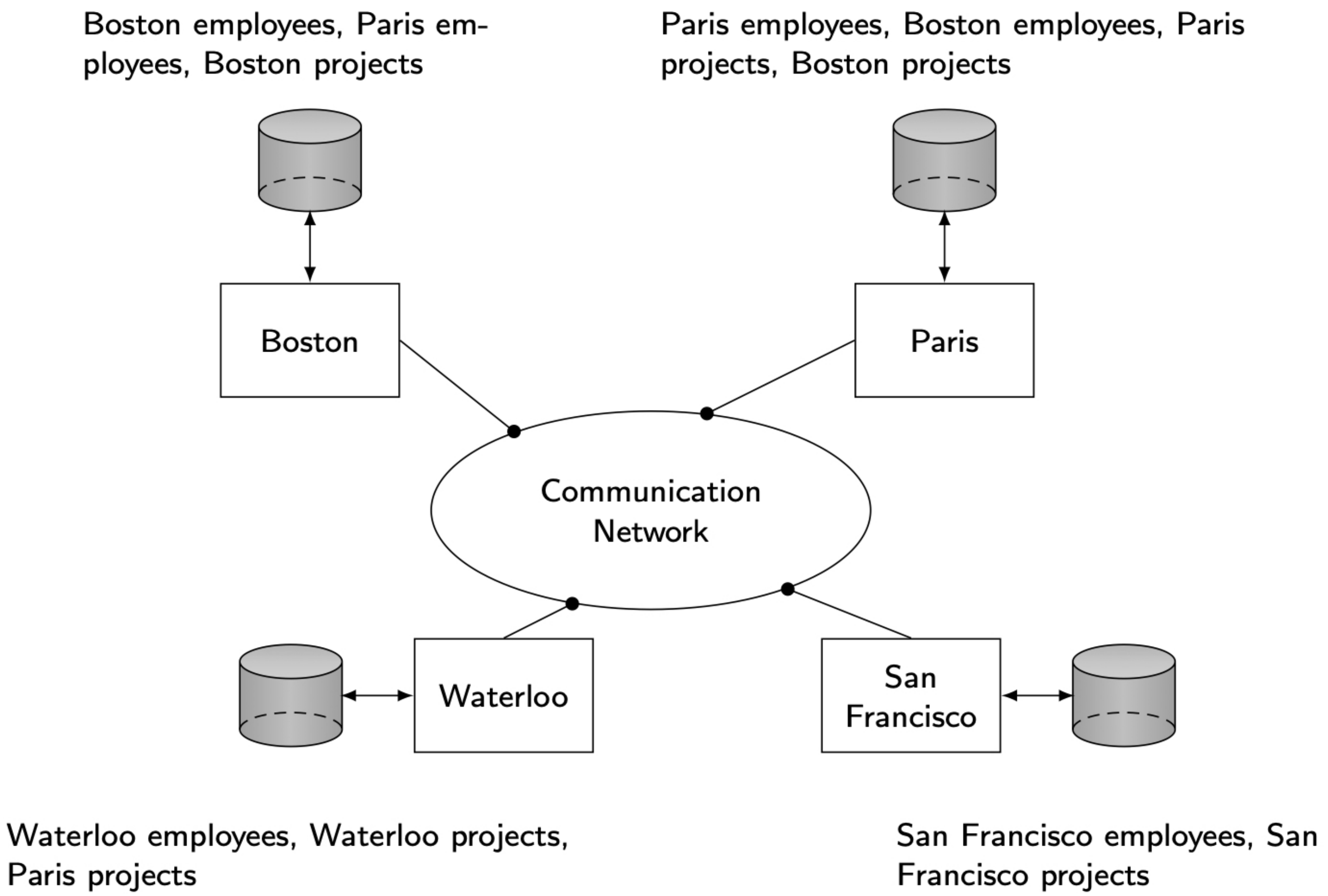


What is a Distributed Database System?

A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network

A distributed database management system (Distributed DBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users

Distributed DBMS Environment


Boston employees, Paris employees, Boston projects
Paris employees, Boston employees, Paris projects, Boston projects
Boston
Paris
Communication Network
Waterloo
San Francisco
Waterloo employees, Waterloo projects, Paris projects
San Francisco employees, San Francisco projects

Implicit Assumptions

- Data stored at a number of sites → each site *logically* consists of a single processor
- Processors at different sites are interconnected by a computer network → not a multiprocessor system
  - Parallel database systems
- Distributed database is a database, not a collection of files → data logically related as exhibited in the users' access patterns
  - Relational data model
- Distributed DBMS is a full-fledged DBMS
  - Not remote file system, not a TP system

Important Point

Logically integrated

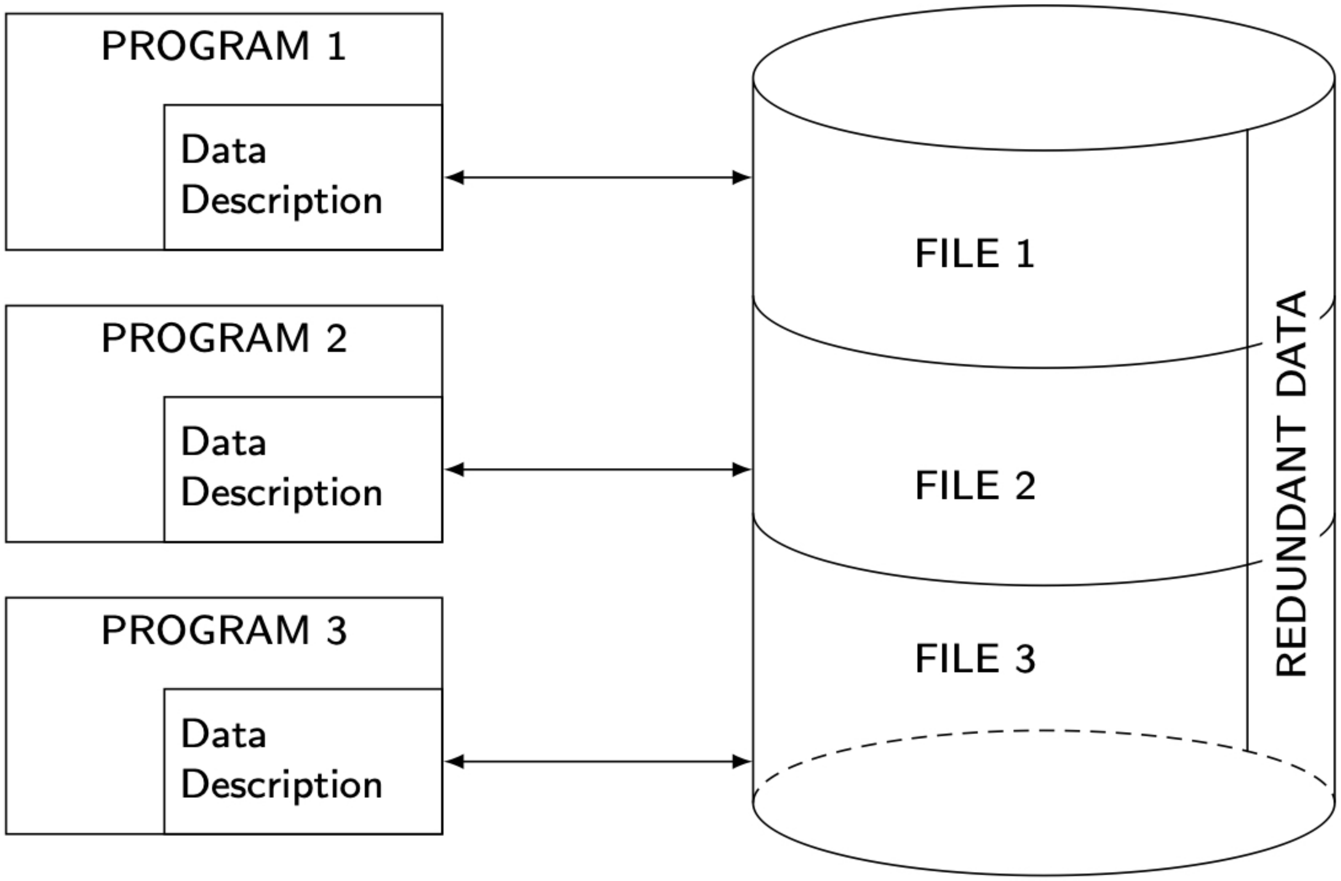
but

Physically distributed

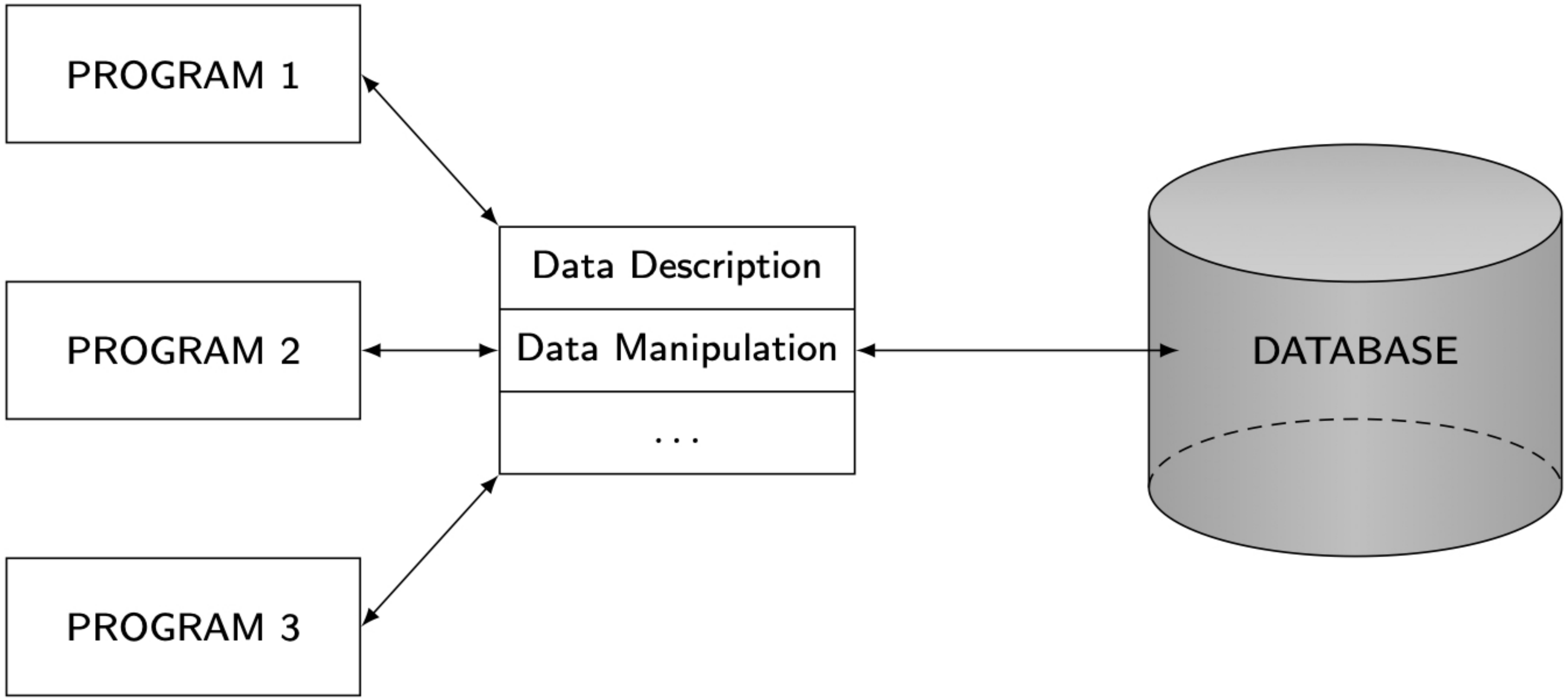
Outline

- Introduction
  - Big data
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - DDBMS issues
  - Distributed DBMS architecture
  - New database systems

History – File Systems

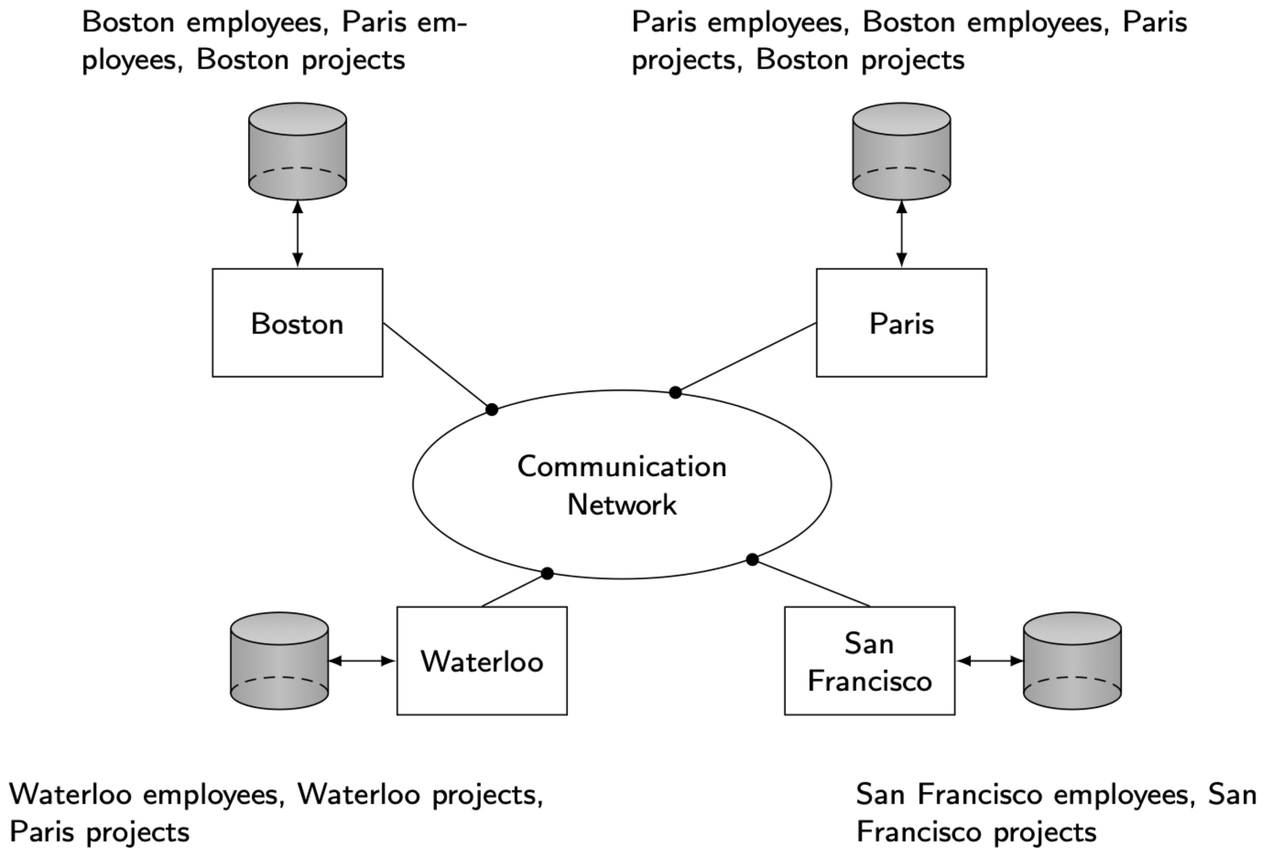


History – Database Management



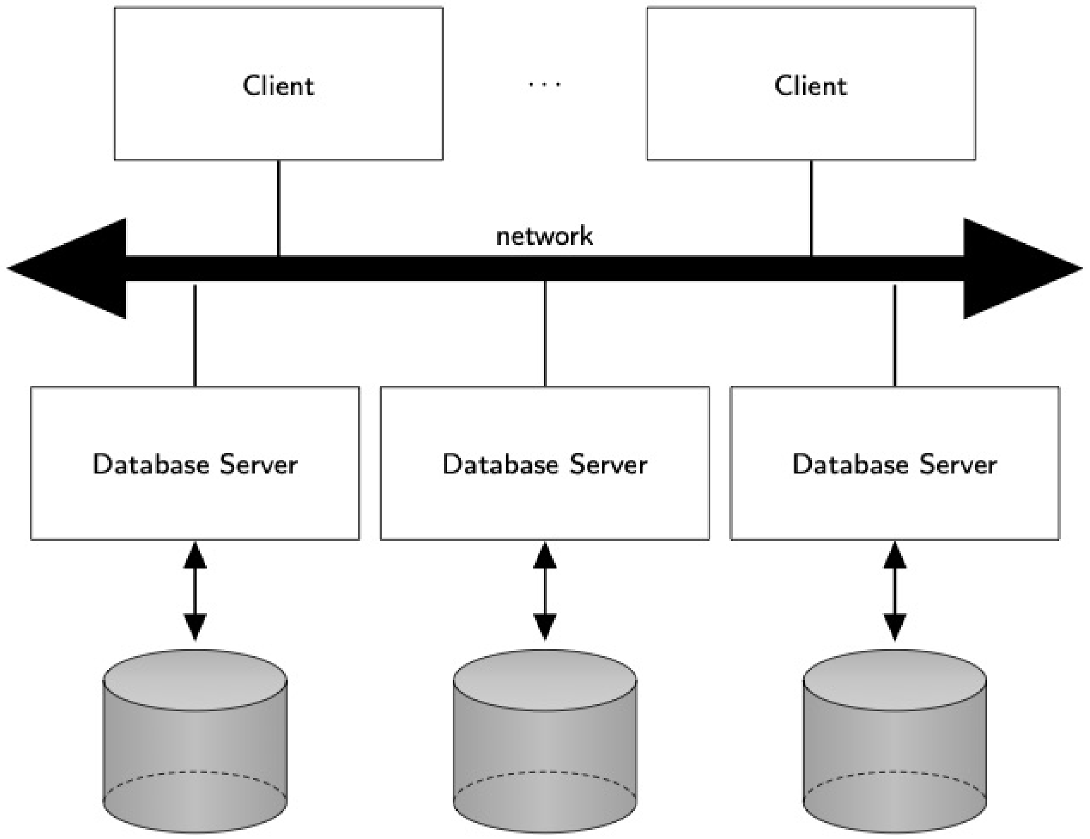
History – Early Distribution

Peer-to-Peer (P2P)

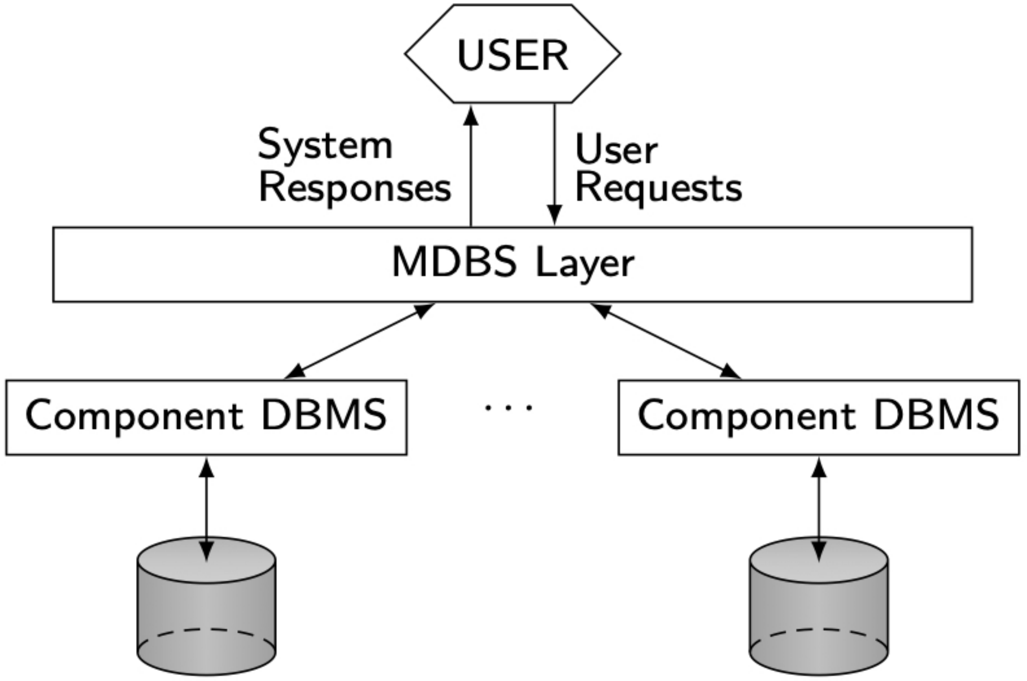


Boston employees, Paris employees, Boston projects
Paris employees, Boston employees, Paris projects, Boston projects
Boston
Paris
Communication Network
Waterloo
San Francisco
Waterloo employees, Waterloo projects, Paris projects
San Francisco employees, San Francisco projects

History – Client/Server



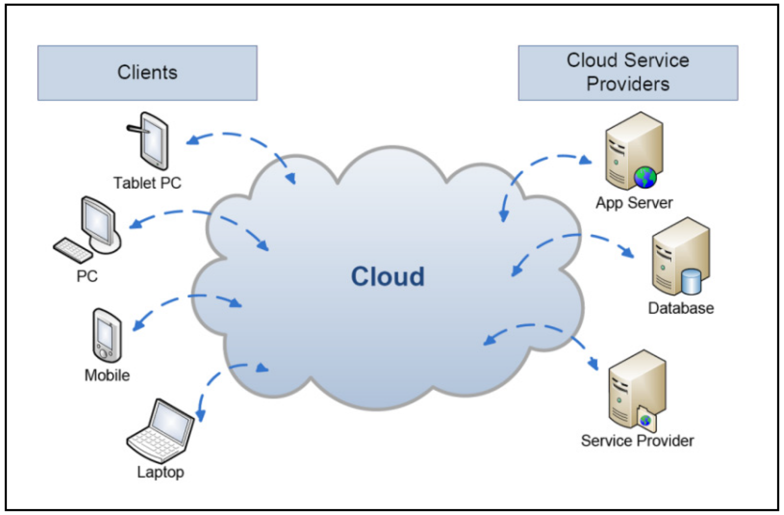
History – Data Integration



History – Cloud Computing

On-demand, reliable services provided over the Internet in a cost-efficient manner

- Cost savings: no need to maintain dedicated compute power
- Elasticity: better adaptivity to changing workload



Data Delivery Alternatives

- Delivery modes
  - Pull-only
  - Push-only
  - Hybrid
- Frequency
  - Periodic
  - Conditional
  - Ad-hoc or irregular
- Communication Methods
  - Unicast
  - One-to-many
- Note: not all combinations make sense

Outline

- Introduction
  - Big data
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - DDBMS issues
  - Distributed DBMS architecture
  - New database systems

Distributed DBMS Promises

❶ Transparent management of distributed, fragmented, and replicated data

❷ Improved reliability/availability through distributed transactions

❸ Improved performance

❹ Easier and more economical system expansion

Transparency

- Transparency is the separation of the higher-level semantics of a system from the lower level implementation issues.
- Fundamental issue is to provide data independence in the distributed environment
  - Network (distribution) transparency
  - Replication transparency
  - Fragmentation transparency
    - horizontal fragmentation: selection
    - vertical fragmentation: projection
    - hybrid

Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E8	P3	Manager	40

PROJ

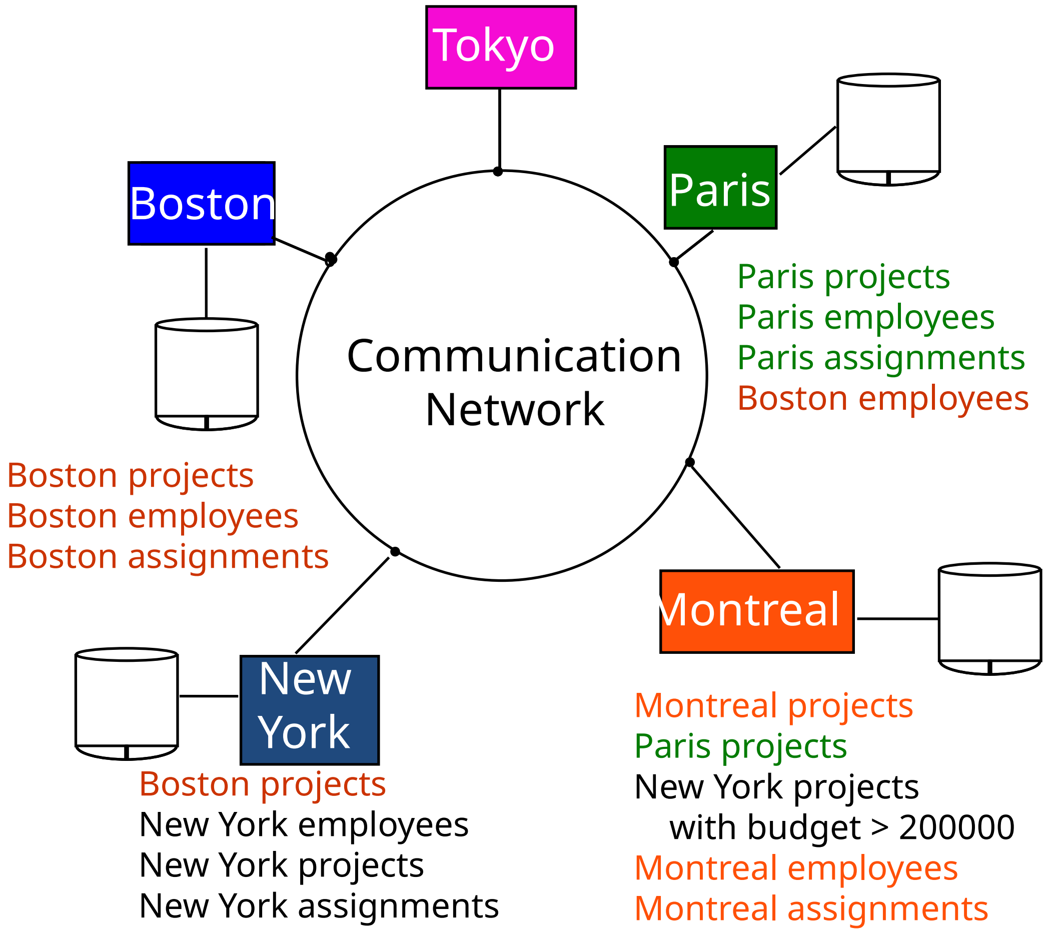
PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

PAY

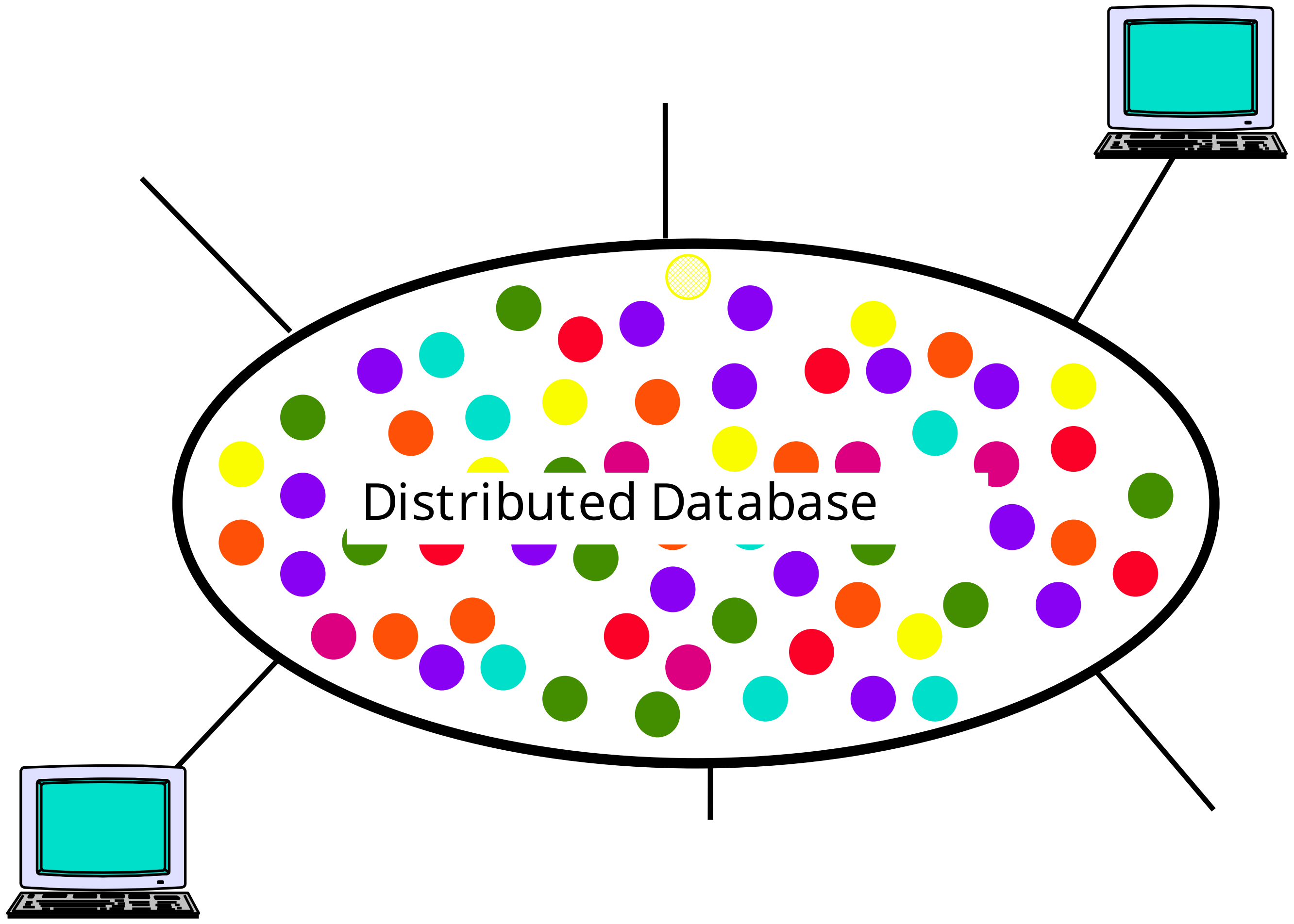
TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

Transparent Access

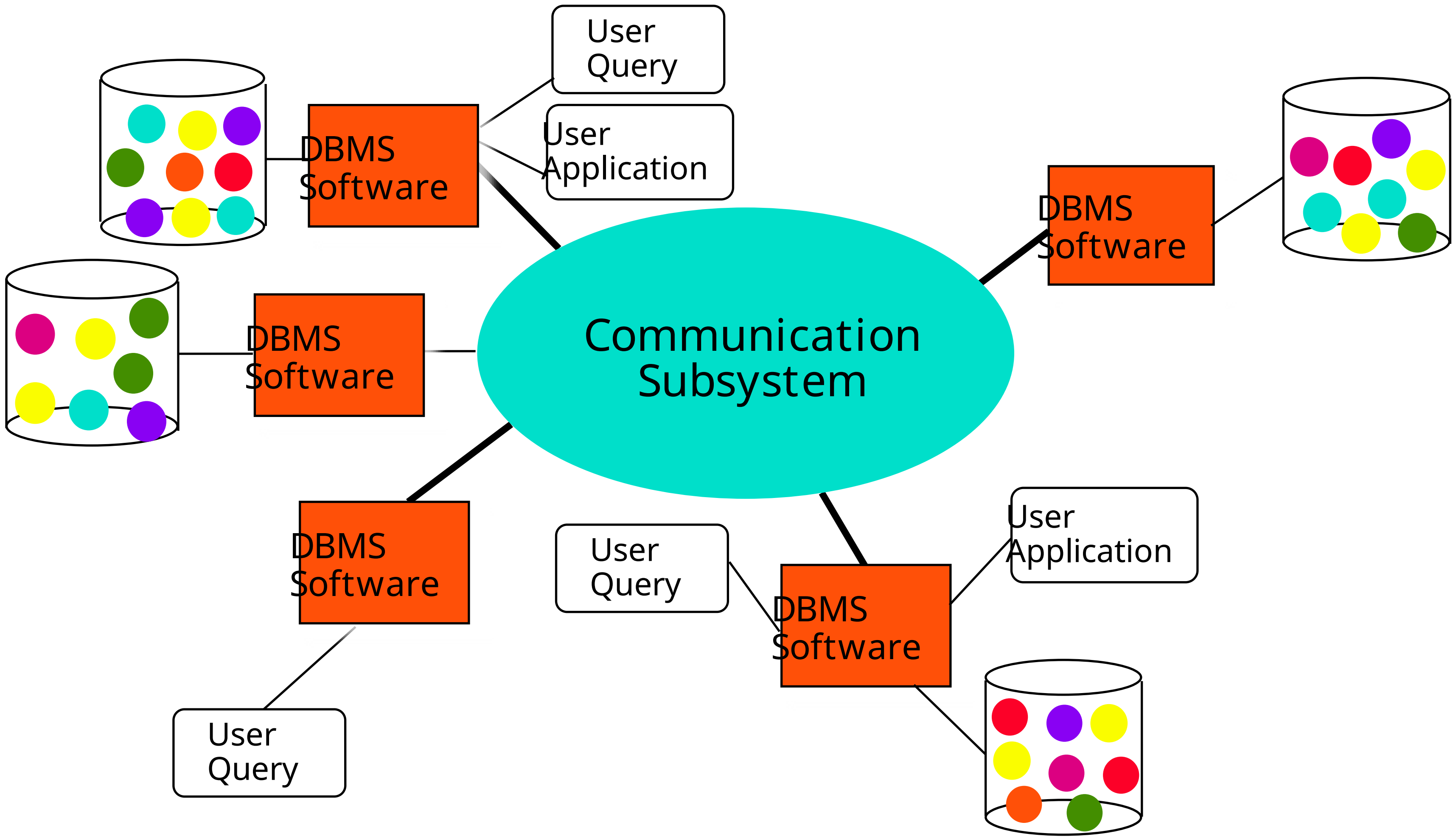
```
SELECT    ENAME,SAL
FROM      EMP,ASG,PAY
WHERE     DUR > 12
AND       EMP.ENO = ASG.ENO
AND       PAY.TITLE =
   EMP.TITLE
```



Distributed Database - User View



Distributed DBMS - Reality



Types of Transparency

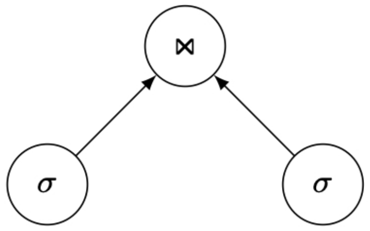
- Data independence
- Network transparency (or distribution transparency)
  - Location transparency
  - Fragmentation transparency
- Fragmentation transparency
- Replication transparency

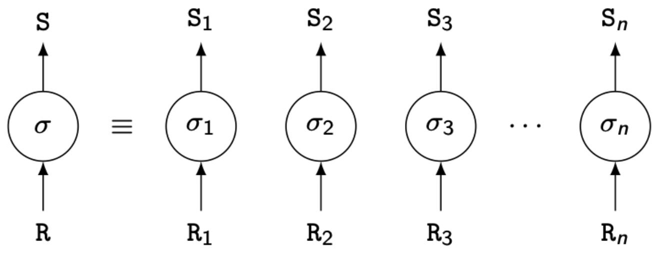
Reliability Through Transactions

- Replicated components and data should make distributed DBMS more reliable.
- Distributed transactions provide
  - Concurrency transparency
  - Failure atomicity
- Distributed transaction support requires implementation of
  - Distributed concurrency control protocols
  - Commit protocols
- Data replication
  - Great for read-intensive workloads, problematic for updates
  - Replication protocols

Improved Performance

- Proximity of data to its points of use
  - Requires some support for fragmentation and replication
- Parallelism in execution
  - Inter-query parallelism
    - Enables the parallel execution of multiple queries
  - Intra-query parallelism
    - Distributed DBMS
      - Splitting a query into parts (each part exec on one site)
    - Parallel DBMS
      - Inter-operator parallelism (Pipelined + Independent)
      - Intra-operator parallelism





Scalability

- Issue is database scaling and workload scaling
- Adding processing and storage power
- Scale-out: add more servers
    - Scale-up: increase the capacity of one server → has limits

Outline

- Introduction
  - Big data
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - DDBMS issues
  - Distributed DBMS architecture
  - New database systems

Distributed DBMS Issues

- Distributed database design
  - How to distribute the database?
  - Units of distribution: fragments, relations
  - Horizontal and vertical fragmentation
    - Partitioning a table horizontally
    - Splitting a table vertically (se later Column stores)
  - Data placement
    - Local query processing – no need to copy tables
    - Parallel execution of query parts
  - Replicated & non-replicated database distribution

Distributed DBMS Issues

- Distributed query processing
  - Origins: System R, Ingres, R*, SDD
  - Exploiting intra-node and inter-node parallelisms
    - Multi-processor and multi-core systems
    - Distributed data nodes (normally shared-nothing systems)
    - Pipelined, data and independent parallelisms
  - Optimization problem (data transmission + local processing)
    - Two phase optimization: global optimization and join ordering
    - Algorithms: Exhausive search, dynamic programming, randimized search, genetic algorithms
  - Query execution
    - Heavy use of indexes; index-only plans
    - Join algorithms: nested loop join, hash joins, associative join, merge-sort join, semi-joins
  - Volcano query optimizer (Graefe, Wisconsin)

Distributed DBMS Issues

- Parallel DBMS
  - Objectives: high scalability and performance
  - Distributed and parallel DBMSs are merging into one area
    - Common hardware, methods and algoritms.
    - Parallel DBMS uses new, usually sophisticated hardware.
  - Parallel system architecture
    - Shared-Memory, Shared-Disk and Shared-Nothing
  - Query processing
    - Exploring parallel processing, new hardware, join processing
    - The pool for join algorithms is the same as for distributed systems.
    - Load-balancing, data partitioning, data placement, multi-processor and multi-core environemt, etc.
  - Cluster computing
    - Autonomous dbms-s interconnected by middleware, often used recently.
  - Paper:
    - Schuh, et.al., Experimental Comparison of Thirteen Relational Equi-Joins in Main Memory, SIGMOD 2016.

Distributed DBMS Issues

- In-memory DBMS
  - There is finally enough RAM
    - First in-memory DBMSs were proposed in 80'.
  - Classical relational systems are based on a buffer manager.
  - In-memory DBMS stores tables in main memory.
    - Different data structures possible (current research)
    - Indexes are in main-memory as well; new proposals
  - New game!
    - Instead of hard disc we now have main memory and 10-100X faster cache.
    - Processor caches have memory of the size 1-120M (L1, L2, L3).
    - Concurrency control remains but there are different bottlenecks
    - Lock trashing, Timestamp allocation, Memory allocation.
    - WAL protocol still needed for recovery
    - New indexes were proposed (BW-tree, LSM tree, Hash tables, Trie index)
  - Paper:
    - Yu, et.al.,Staring into the Abyss: An Evaluation of CC with One Thousand Cores, VLDB 2014.

Distributed DBMS Issues

- Concurrency Control (CC)
  - Synchronization of concurrent transactions
    - Transaction can work in parallel (in some cases)
  - Protocols and techniques of CC
  - 2PC protocol
    - Problem in distributed environment
  - Timestamp ordering
  - Multi-version CC
    - Very popular recently (e.g., Postgres)
  - Optimistic CC
  - Snapshot Isolation (SI)
  - Deadlock management

Distributed DBMS Issues

- Synchronization and coordination
  - Novel approaches: Quorums
    - Majority of nodes in a group required for a decision (not all!).
  - Consensus on a decision (Paxos, Raft)
    - Tasks: Replicated write, leader election, etc.
    - Paxos used in F1 (Google)
  - Coordination of a system (ZooKeeper, Yahoo)
    - Coordination primitives
    - Similar to Chubby in Bigtable (Google)
    - Tasks; Linearization of requests, ordering events, Lock manager, etc.
    - Used in Apache Hadoop, Accumulo, HBase, Hive, Spark, Druid, etc.
  - Paper:
    - Wu, et.al., Empirical Evaluation of In-Memory MVCC, VLDB2017

Distributed DBMS Issues

- Replication
  - Mutual consistency: strict and weak
    - Eventual consistency
  - Eager vs lazy & Centralized vs distributed
  - Eager-centralized
    - Classical 2PL protocol, slow, no inconsistencies, no need for coordination
  - Eager-distributed
    - No inconsistencies, elegant (symmetrical), slow, updates coordinated
  - Lazy-centralized
    - Short response time, local copy out of date, inconsistencies, no coordination
  - Lazy-distributed
    - Shortest response time, inconsistencies, lost updates, no coordination

Distributed DBMS Issues

- Availability
  - Basic relational distributed 2PC breaks availability
    - No way to replicate the data
  - Fault tolerance required
    - What to do when 2PC participant, Coordination leader, ... is not available?
    - Some solutions presented (e.g., Dynamo)
  - Network partitions
    - What to do when the network is partitioned?

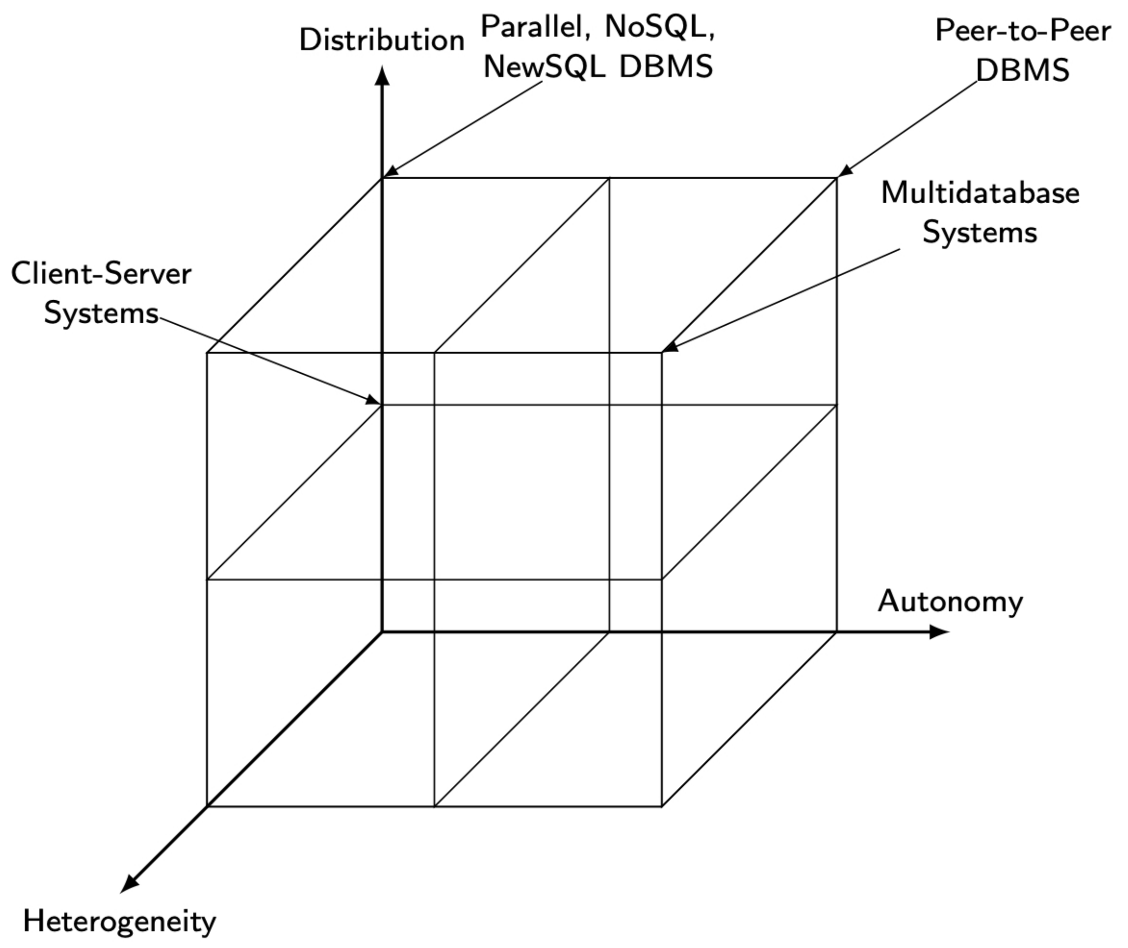
Distributed DBMS Issues

- Big data processing
  - 4V: volume, variety, velocity, veracity
  - MapReduce & Spark
  - Stream data
  - Graph analytics
  - NoSQL
  - NewSQL
  - Polystores
- Presented in the second part of course!

Outline

- Introduction
  - Big data
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - Design issues
  - Distributed DBMS architecture
  - New database systems

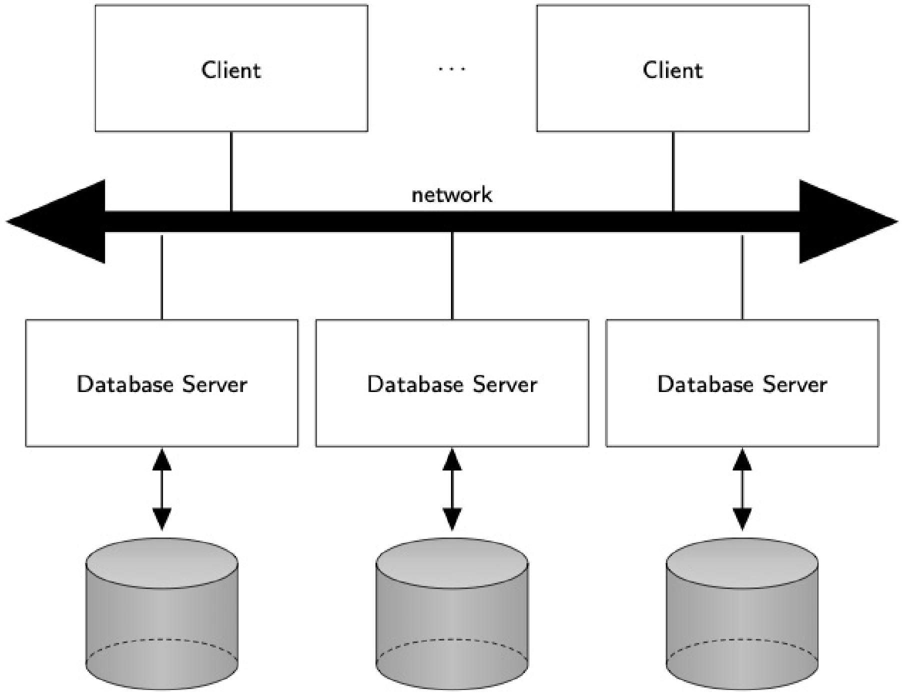
DBMS Implementation Alternatives

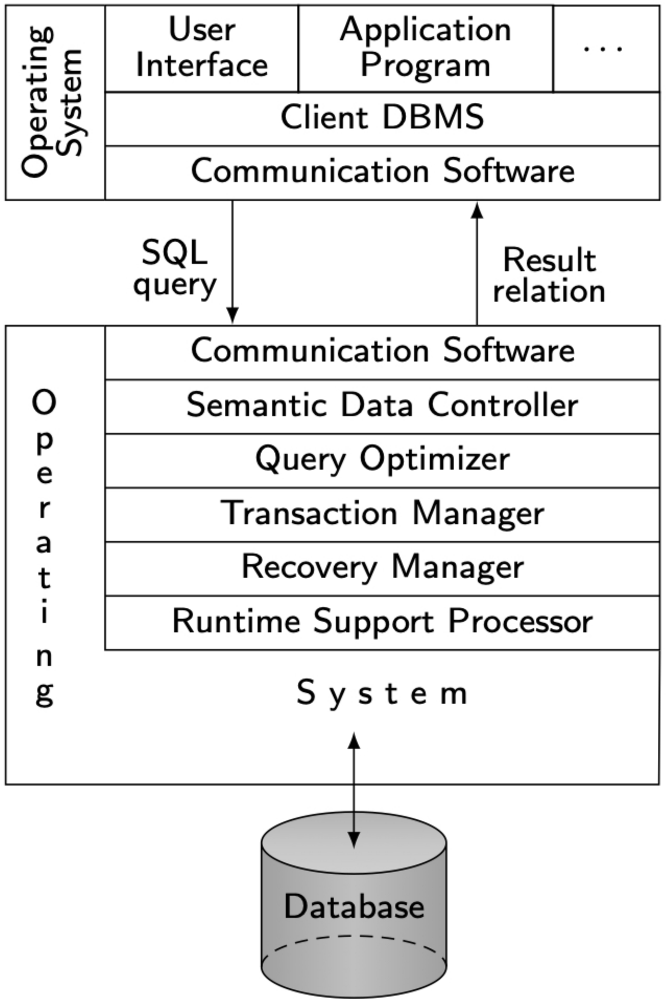


Dimensions of the Problem

- Distribution
  - Whether components of the system are located on the same machine or not
- Heterogeneity
  - Various levels (hardware, communications, operating system)
  - DBMS important one
    - data model, query language,transaction management algorithms
- Autonomy
  - Types: Tight integration, semi-autonomous, total isolation
  - Various versions
    - Design autonomy: Ability of a component DBMS to decide on issues related to its own design.
    - Communication autonomy: Ability of a component DBMS to decide whether and how to communicate with other DBMSs.
    - Execution autonomy: Ability of a component DBMS to execute local operations in any manner it wants to.

Client/Server Architecture

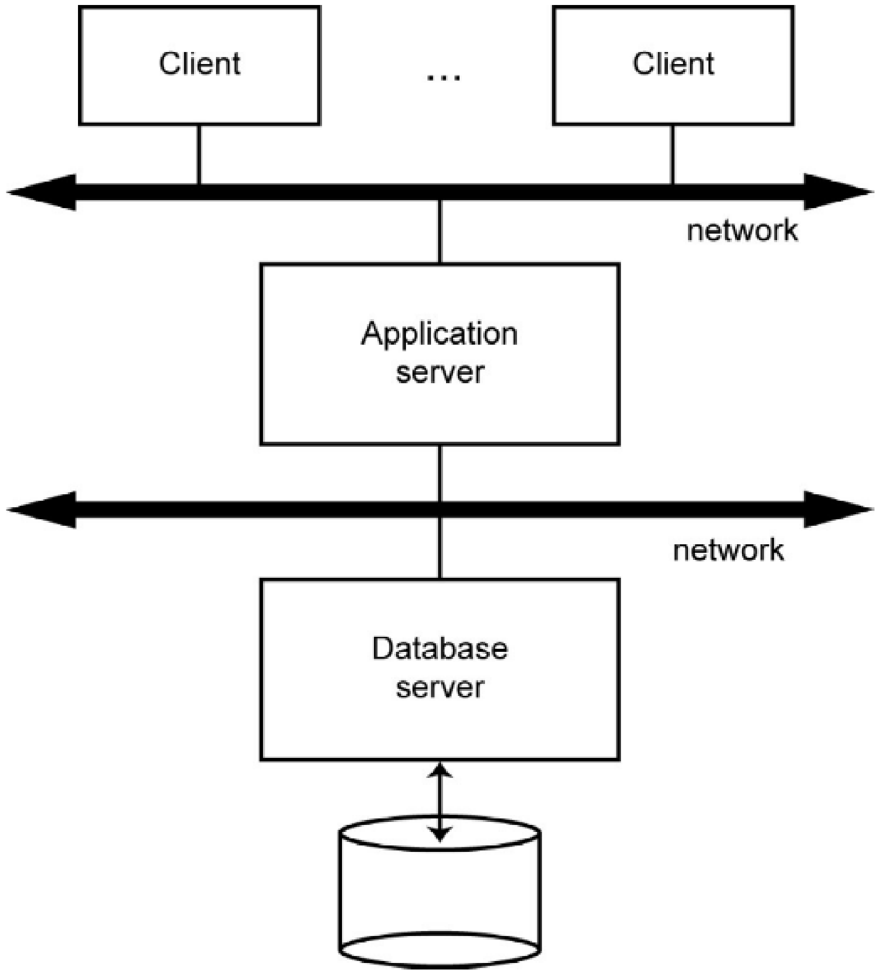




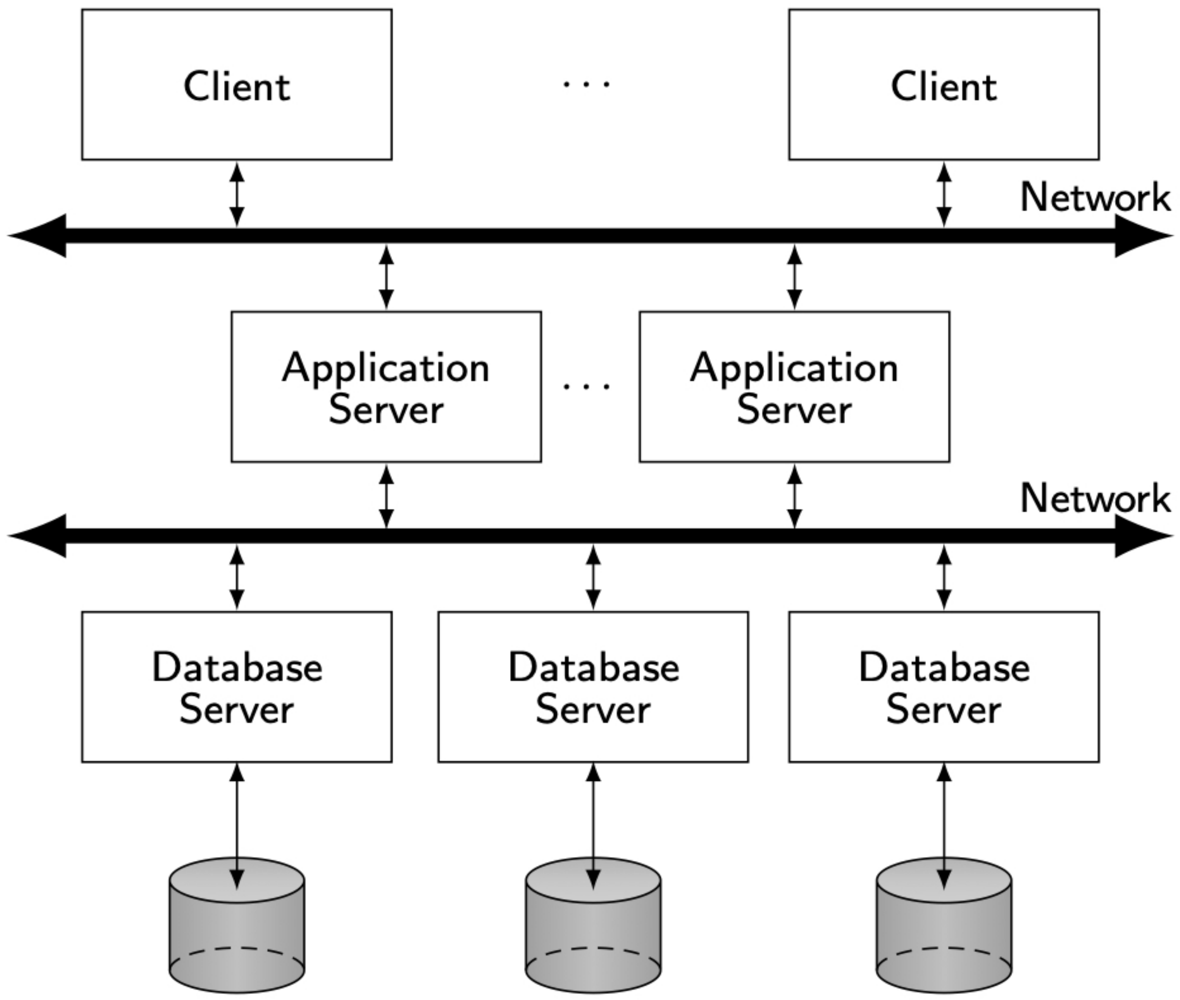
Advantages of Client-Server Architectures

- More efficient division of labor
- Horizontal and vertical scaling of resources
- Better price/performance on client machines
- Ability to use familiar tools on client machines
- Client access to remote data (via standards)
- Full DBMS functionality provided to client workstations
- Overall better system price/performance

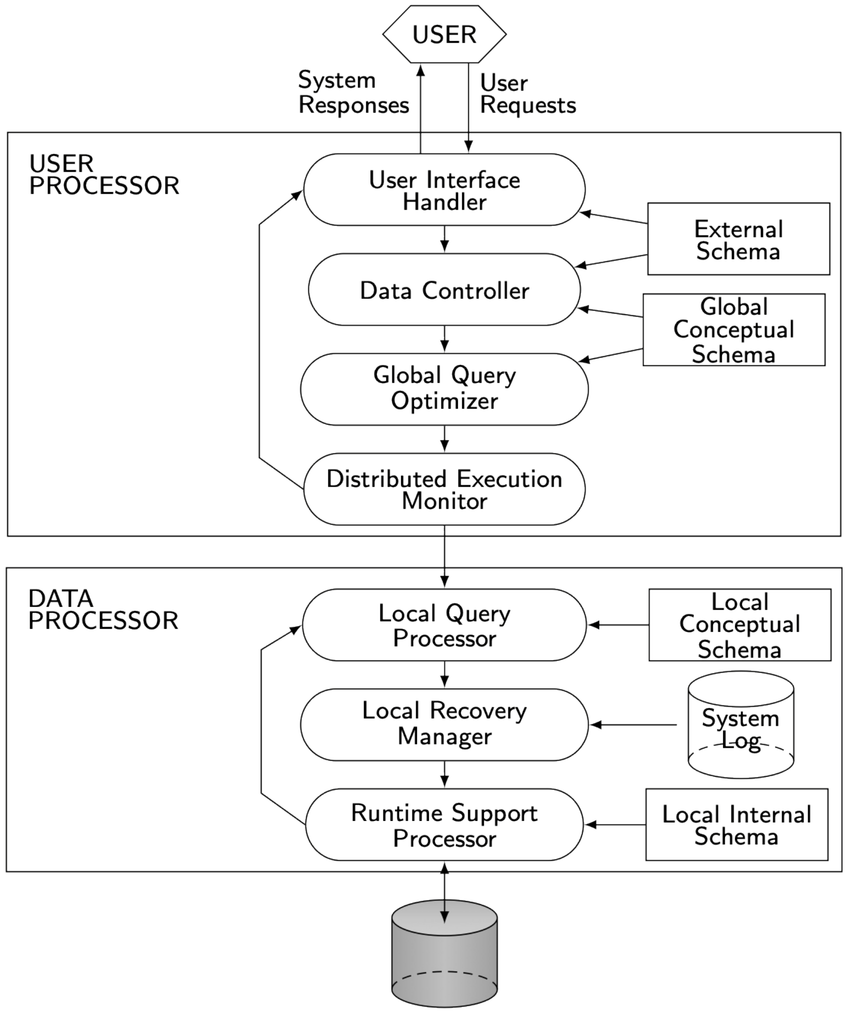
Database Server



Distributed Database Servers



Peer-to-Peer Component Architecture



Peer-to-Peer Component Architecture

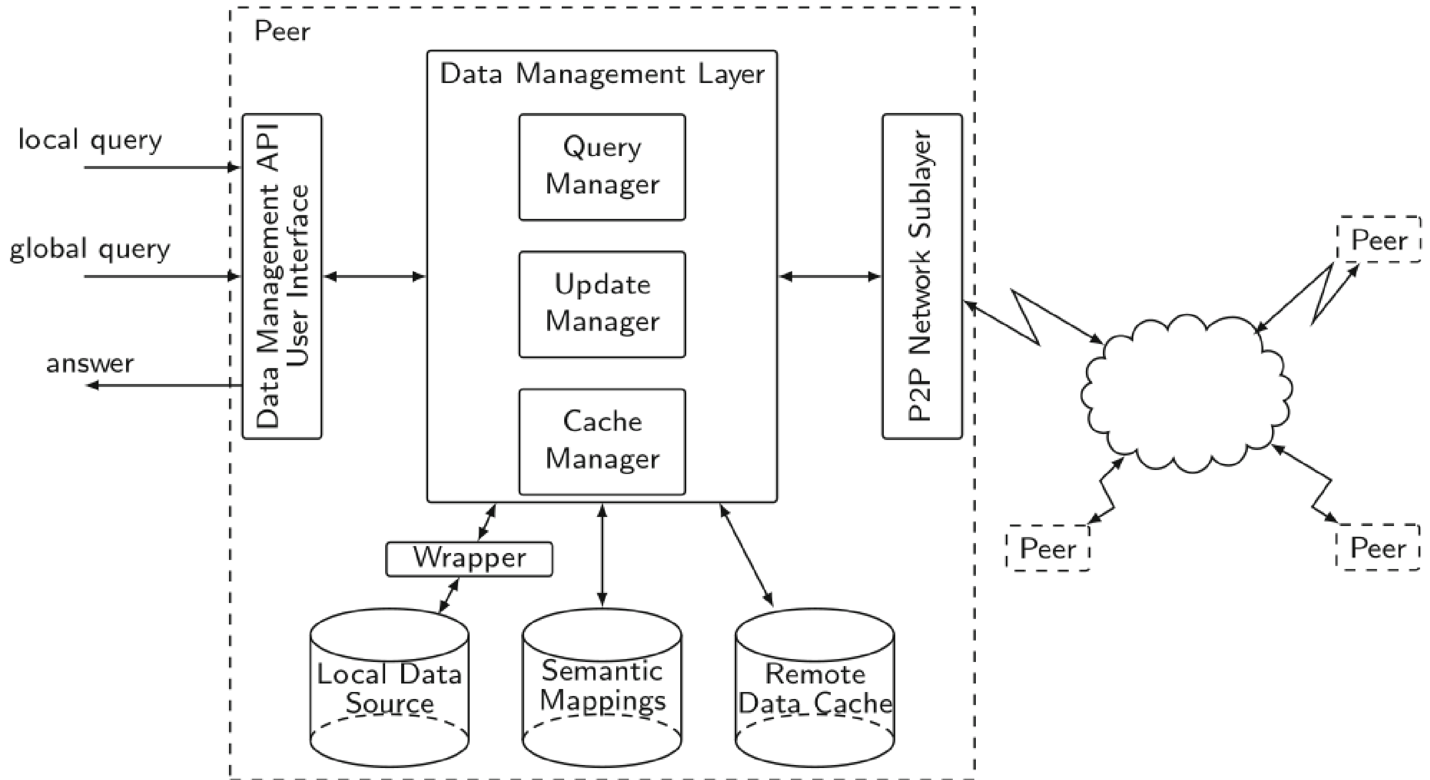
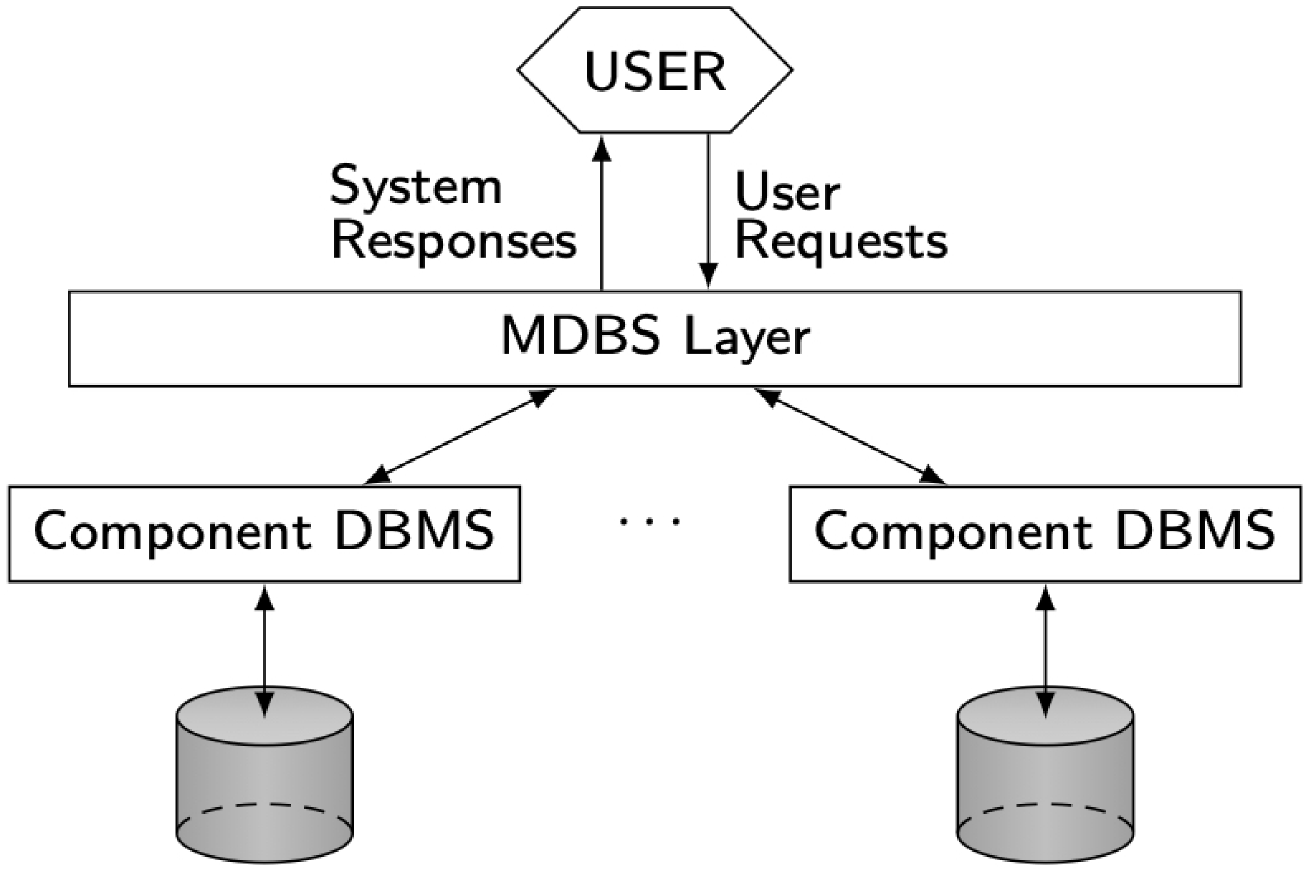
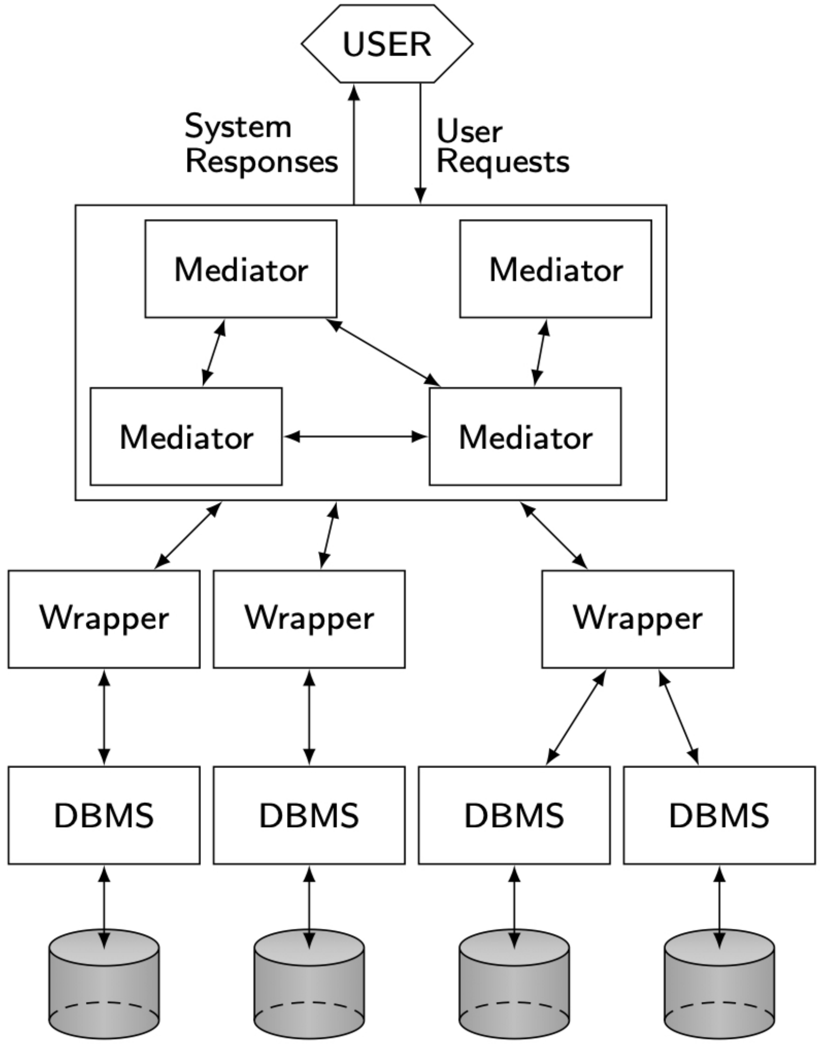


Fig. 9.1 Peer reference architecture

MDBS Components & Execution



Mediator/Wrapper Architecture

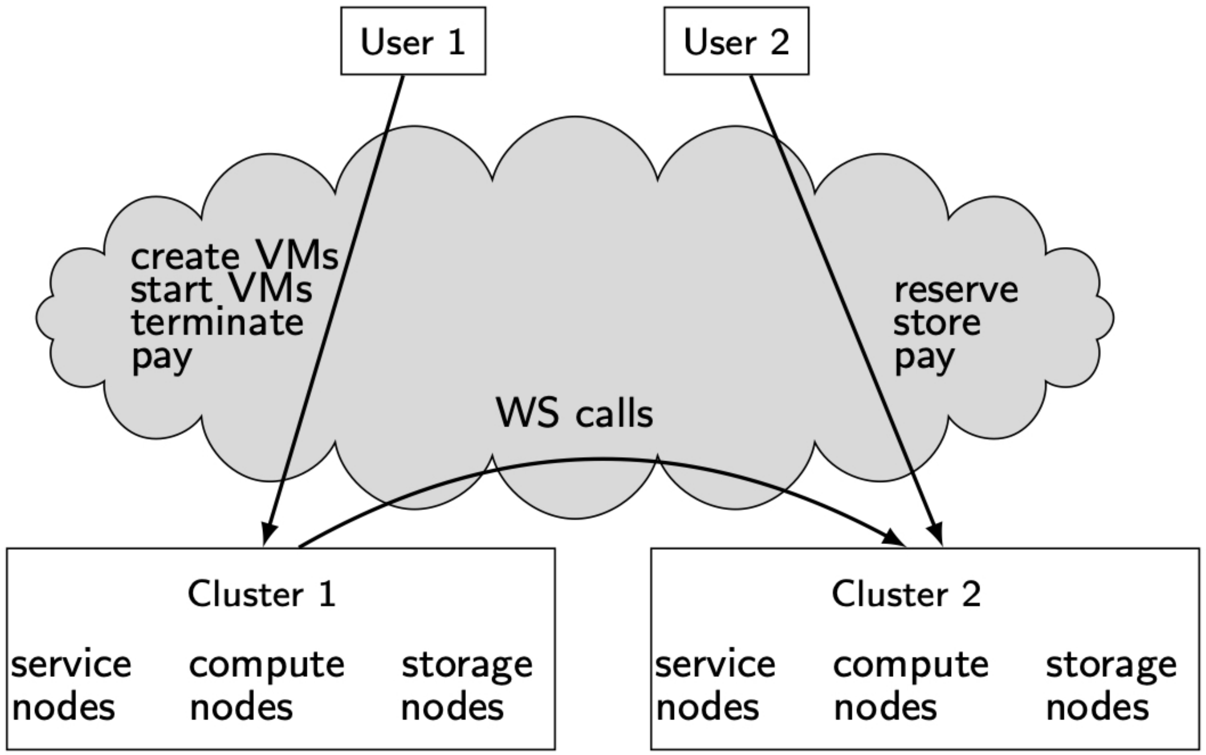


Cloud Computing

On-demand, reliable services provided over the Internet in a cost-efficient manner

- IaaS – Infrastructure-as-a-Service
- PaaS – Platform-as-a-Service
- SaaS – Software-as-a-Service
- DaaS – Database-as-a-Service

Simplified Cloud Architecture

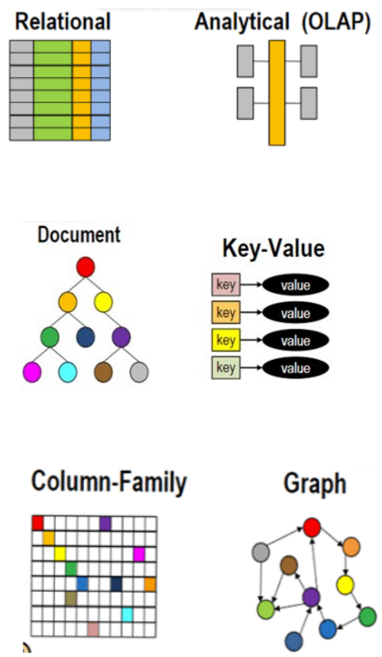


Outline

- Introduction
  - Big data
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - Design issues
  - Distributed DBMS architecture
  - New database systems

New DBMSs and Big Data Processing

- Key-Value stores
- Document stores
- Column-oriented DBMS
- Graph database systems
- NewSQL DDBMS
- Map-Reduce systems
- Data-flow systems
- Stream query processing



The End of an Architectural Era

- Paper:
  - Stonebraker, et.al, The End of an Architectural Era (It's Time for a Complete Rewrite), VLDB 2007.
- Michael Stonebraker, UCB
  - Current DBMSs: "one size fits all" solution, in fact, excel at nothing"
  - H-Store developed at the M.I.T. beats up RDBMSs by nearly two orders of magnitude in the TPC-C benchmark (see commercialization VaultDB)
  - RDBMSs" are 25 year old legacy code lines that should be retired in favor of a collection of "from scratch" specialized engines.
    - Code lines and architectures designed for yesterday's needs"
  - Popular relational DBMSs all trace their roots to System R from the 1970s
    - IBM's DB2 is a direct descendant of System R,
    - Microsoft's SQL Server has evolved from Sybase System 5 (another direct System R descendant) and
    - Oracle implemented System R's user interface in its first release.

Design considerations

- Yesterday’s vs. Today’s Needs
  - Movements in Programming Languages and Development Frameworks
  - Large Main Memory available
  - Multi-Threading and Resource Control
  - Grid Computing and Fork-Lift Upgrades
  - High Availability needed!
  - Horizontal Scalability and Running on Commodity Hardware
  - Shared-nothing support at the bottom of the system
  - No Knobs
    - Current RDBMSs were designed in an era, when computers were expensive and people were cheap. Today we have the reverse.equirements of Cloud Computing

Design Considerations

- High Throughput and Scalability
    - Complexity and Cost of Setting up Database Clusters
    - Myth of Effortless Distribution and Partitioning of Centralized Data Models
    - Most data can be stored in Main Memory (see new caches)
    - Multi-Threading can be used effectively
    - Systems need to be Built from Scratch with Scalability in Mind

Design Considerations

- Unneeded Complexity and Performance Bottlenecks
  - Avoidance of Expensive Object-Relational Mapping
  - Persistent redo-logs have to be avoided when possible
  - JDBC/ODBC-like interfaces
  - Eliminate an undo-log wherever practical
  - Dynamic locking to allow concurrent access
  - Multi-threaded datastructures lead to latching of transactions
  - Two-phase-commit (2PC) transactions should be avoided whenever possible

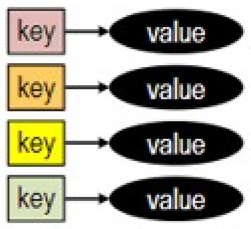
Design Considerations

- Covering simple types of transactions
  - Tree Schemes
    - 1-n relationship with its ancestor require joins
    - The schema is a tree of 1-n relationships
    - Equality predicates on the primary key(s) of the root node
  - Single-Sited Transactions
  - One-Shot Transactions
  - Two-Phase Transactions
    - Strongly Two-Phase Transactions
  - Transaction Commutativity
    - Sterile Transactions Classes

Consequences

- We are heading toward a world with at least 5 specialized engines
  - Death of the “one size fits all” legacy systems
    - 1970s: DBMS world contained only business data processing applications
- Areas which need specialized DBMSs
  - Data warehouses, Big data, Internet data, Text, Scientific data, Semi-structured data, Graphs, Streams, etc.

Key-/Value-Stores

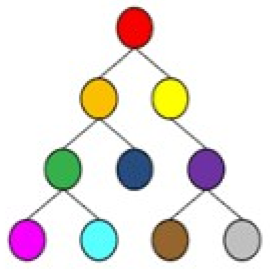


- A simple, common data model:
  - a map/dictionary, allowing clients to put and request values per key.
- Modern key-value stores favor high scalability over consistency
- Most of them also omit rich ad-hoc querying and analytic features
  - Especially joins and aggregate operations are set aside
- Key-/value-stores have existed for a long time
  - e.g. Berkeley DB

Key-/Value-Stores

- Examples of systems
  - Key-value cache
    - Memcached, Coherence (Oracle), Velocity, Repcached, ElastiCache,
    - Infinispan, Jboss Cache, Aerospike
  - Key-Value Store
    - Dynamo, Voldemort, Dynomite, Riak, Redis, RAMCloud, LevelDB

Document stores



- Data model
  - Documents
    - Self-describing
    - Hierarchical tree structures (JSON, XML, …)
      - Scalar values, maps, lists, sets, nested documents, …
    - Identified by a unique identifier (key, …)
  - Documents are organized into collections
- Query patterns
  - Create, update or remove a document
  - Retrieve documents according to complex query conditions
- Observation
  - Extended key-value stores where the value part is examinable!

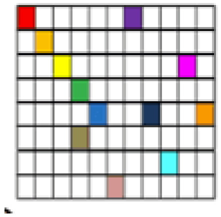
Document stores

- Suitable use cases
  - Event logging, content management systems, blogs, web analytics, e-commerce applications, …
    - i.e. for structured documents with similar schema
- When not to use
  - Set operations involving multiple documents
  - Design of document structure is constantly changing
    - i.e. when the required level of granularity would outbalance the advantages of aggregates

Document stores

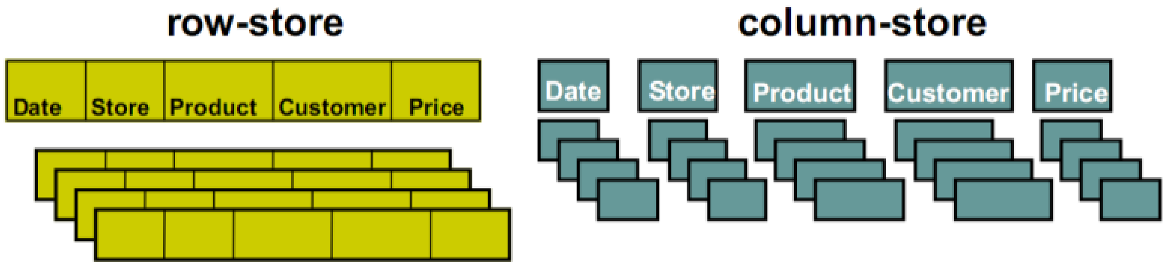
- Representatives
  - MongoDB
  - Couchbase
  - CouchDB
  - RavenDB
  - Terrastore
  - Multi-model:
    - MarkLogic
    - OrientDB
    - OpenLink Virtuoso
    - ArangoDB

Column-Oriented Databases

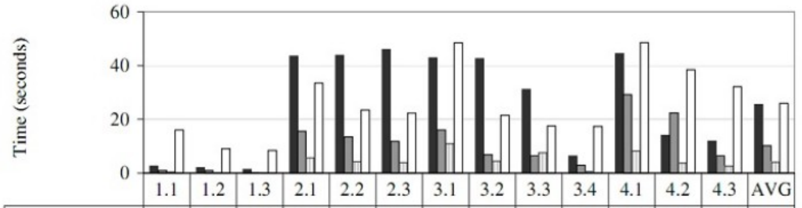


- The approach to store and process data by column instead of row
  - Origin in analytics and business intelligence
    - Column-stores operating in a shared-nothing massively parallel processing architecture can be used to build high-performance applications
- Column-orientation has a number of advantages
  - One column is always accessed (not whole table of records)
  - An index on a column is a representation of column
  - Scalability of the column-oriented database
- Puristic column-oriented stores
  - Sybase IQ
  - Vertica
  - C-store

Column-Oriented Databases



- Column store features
  - Index-only plans, heavy compression, late materialization, block iteration,
- Column stores outperform commercial row-oriented DBs
  - Daniel Abadi,



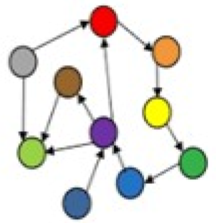
	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	3.4	4.1	4.2	4.3	AVG
■ RS	2.7	2.0	1.5	43.8	44.1	46.0	43.0	42.8	31.2	6.5	44.4	14.1	12.2	25.7
▩ RS (MV)	1.0	1.0	0.2	15.5	13.5	11.8	16.1	6.9	6.4	3.0	29.2	22.4	6.4	10.2
▥ CS	0.4	0.1	0.1	5.7	4.2	3.9	11.0	4.4	7.6	0.6	8.2	3.7	2.6	4.0
□ CS (Row-MV)	16.0	9.1	8.4	33.5	23.5	22.3	48.5	21.5	17.6	17.4	48.6	38.4	32.1	25.9

Baseline performance of C-Store "CS" and System X "RS", compared with materialized view cases on the same systems.

Column-Oriented Databases

- Less puristic column stores subsume datastores that integrate column- and row-orientation
  - Bigtable (Google) based on GFS
  - Hypertable based on HDFS (Hadoop file system)
  - Hstore also based on HDFS
  - Cassandra Derived from Bigtable and Dynamo

Graph database systems



- Graph data model
  - Data is represented in the form of the graph
  - Any representation can be converted to a graph representation
- Graph representations
  - Adjacency lists, adjecency matrix, triples and triple tables, special data structures
    - Indexes, bitmaps, signature trees, …
  - RDF data model
    - Many levels of representation: data, schema, logic

Graph database systems

- Declarative query language
  - Initially in-memory systems
  - SPARQL query language
    - Data and knowledge query language (RDF inference)
  - Heavy use of indexing
    - Special new index structures
  - Query optimization
    - Dynamic programming, pipelines, bushy trees
  - Distributed databases and query processing

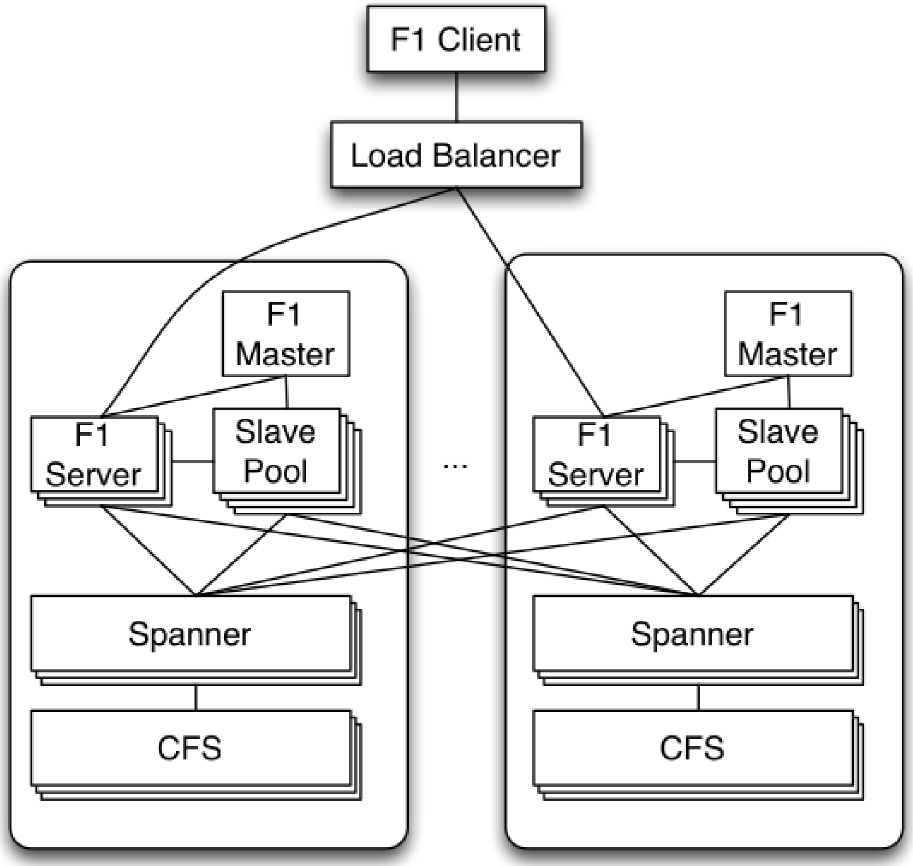
Graph database systems

- Example Graph DBMSs
  - RDF-3X
  - Neo4j
  - Virtuoso
  - ArangoDB
  - OrientDB
  - Dgraph
  - GraphDB
  - Neptune (Amazon)
  - Titan
  - IBM Graph
  - Oracle Graph
  - ...

New relational DDBMS

- Google F1, 2013 (Megastore. 2011)
  - F1 is a fault-tolerant globally-distributed DBMS
  - Storage of Google's AdWords system
  - Genetics: Filial 1 hybrid
  - Combining best aspects of traditional RDBMS and scalable NoSQL systems
- The key goals of F1's design
  - Scalability, availability (never go down), consistency (ACID), usability (full SQL+expected)
  - These design goals were considered to be mutually exclusive
- F1 is built on top of Spanner
  - Scalable data storage, synchronous replication, and strong consistency and ordering properties.

New relational DDBMS



Big Data Analytics

- Map-Reduce systems
- Stream query processing
- Data-flow systems

Map-Reduce Systems

- Brought up by Google employees in 2004
- Task split into two stages:
  - Map:
    - a coordinater designates pieces of data to process a number of nodes which execute a given map function and produce intermediate output.
  - Reduce:
    - the intermediate output is processed by a number of machines executing a given reduce function whose purpose it is to create the final output from the intermediate results, e. g. by some aggregation
- Map and Reduce computation model
  - Map-Reduce is a programming technique
  - Have to be understood in a real functional manner
  - It is used for programming streams
- Restricted to the Map-Reduce model of computation

Map-Reduce Systems

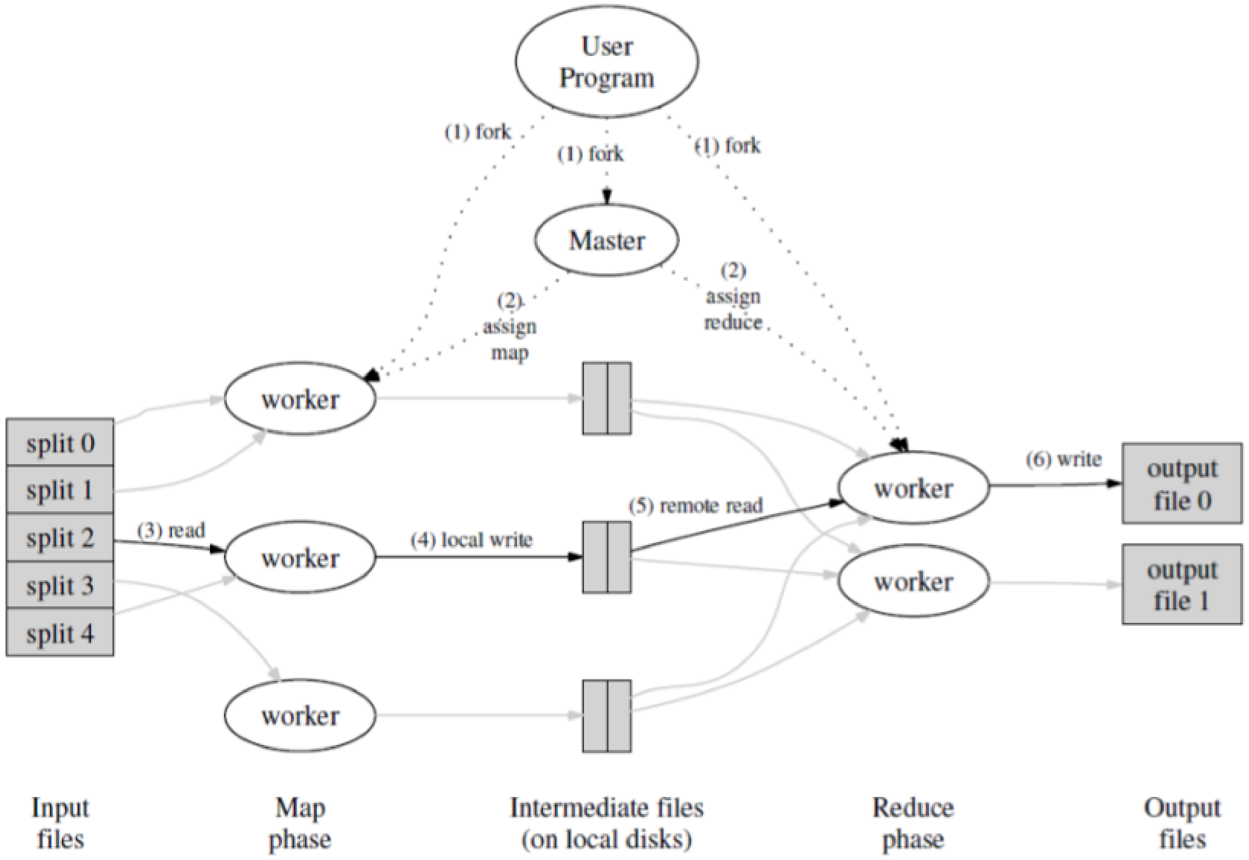


Figure 3.18.: MapReduce – Execution Overview (taken from [DG04, p. 3])

Map-Reduce Systems

- MapReduce paradigm has been adopted by
  - Programming languages (e. g. Python)
  - Frameworks (e.g. Apache Hadoop)
  - NoSQL databases (e. g. CouchDB)
  - Even JavaScript toolkits (e. g. Dojo)

Spark

- Addresses MapReduce shortcomings
- Data sharing abstraction:
  - Resilient Distributed Dataset (RDD)
- Computation model:

1) Cache working set (i.e. RDDs) so no writing-to/reading-from HDFS

2) Assign partitions to the same machine across iterations

3) Maintain lineage for fault-tolerance

Stream data management

- Stream is an append-only sequence of timestamped items that arrive in some order
  - Unbounded stream
  - Typical arrival: <timestamp, payload>
    - Records, triples, structured texts, ...
- Processing models
  - Continuous = arrival is processed as soon as received in the system
    - Apache Storm, Heron
  - Windowed = arrivals are batched in windows, executed in batch
    - Aurora, STREAM, Spark Streaming

Stream data management

- Stream Query Models
  - Persistent queries
  - Push-based (data-driven)
  - Monotonic: result set always grows, output is continuous
  - Non-monotonic: some answers in the result set become invalid with new arrivals, re-computation of the result set
- Stream Query Languages
  - Declarative: SQL-like QLs; CQL, GSQL, ...
  - Procedural: an acyclic graph of operators; Aurora
  - Windowed: Windowed languages; size, slide, …
  - Stateless and Statefull (blocking) operators

Dataflow systems

- Application domain
  - Data-intensive analytics is moving towards complex data-processing tasks such as statistical modeling, graph analysis, machine learning, and scientific computing
- Computation model
  - MapReduce model is restricted
    - Dataflow systems are extending the MapReduce framework with a more generalized dataflow-based execution model
    - new primitive operations in addition to Map and Reduce
    - Systems: Spark, Hyracks, and Nephele
  - Dataflow model can express a wide range of data access and communication patterns
  - Various dataflow-based execution models have been proposed
    - directed acyclic graphs in Dryad,
    - serving trees in Dremel, and
    - bulk synchronous parallel processing in Pregel

Dataflow systems

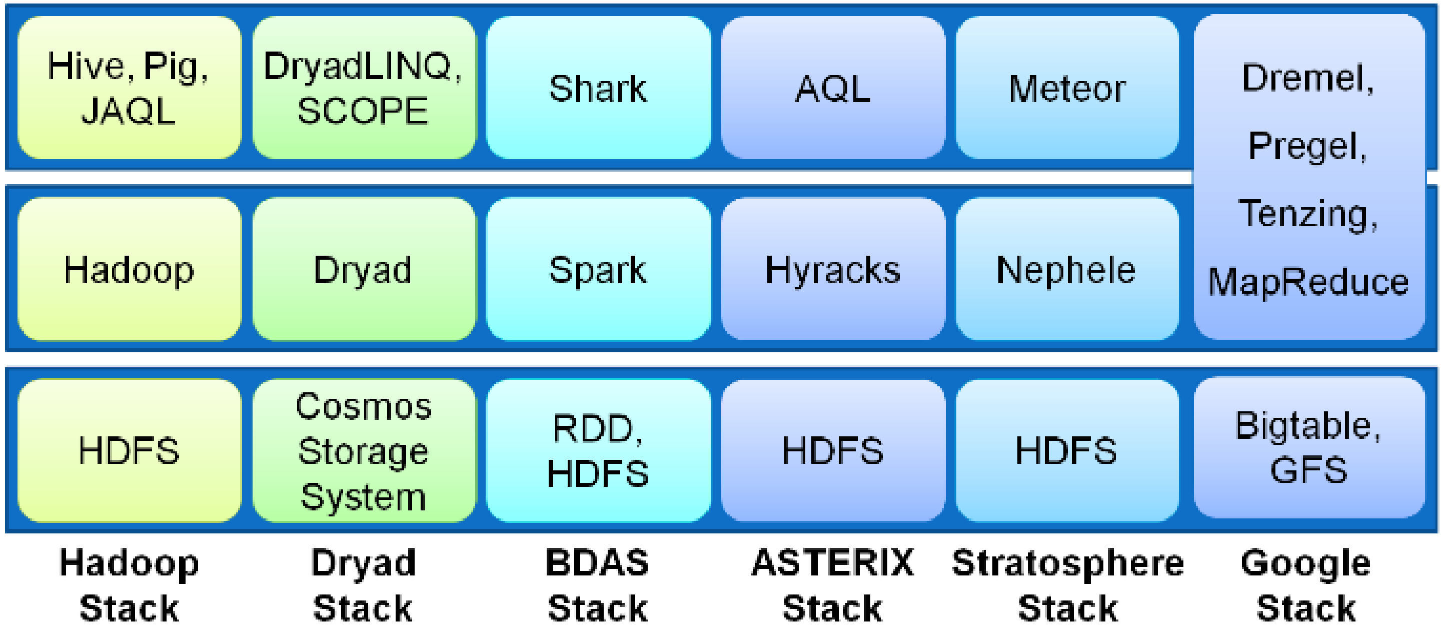


Figure 5.1: Typical dataflow software stacks along with the Hadoop stack.