Advanced Database Systems for Big Data - Challenges

Introduction to Stream Processing

Vera Goebel

Thomas Plagemann

Report Series on Database Research

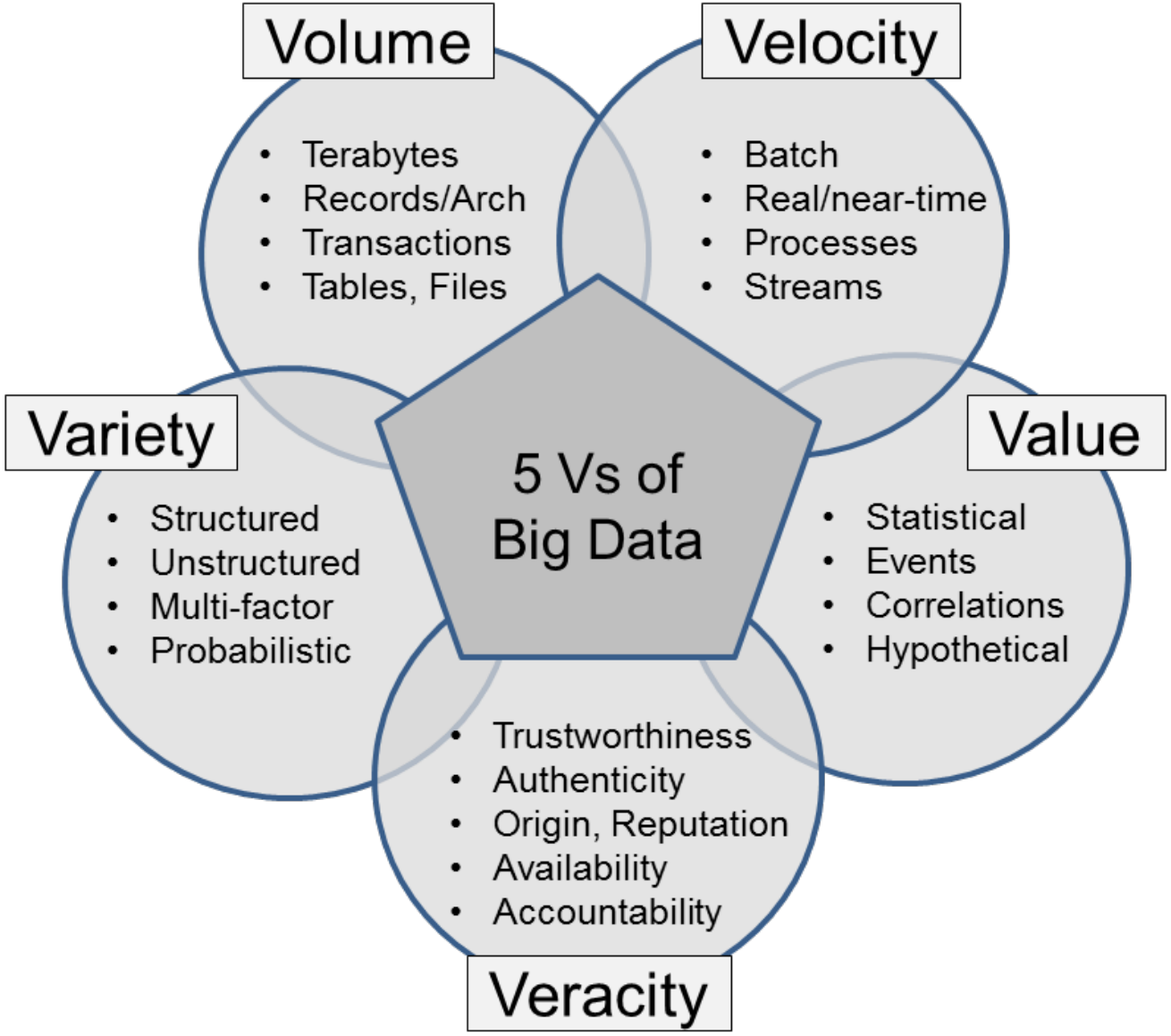
- Leading (30+) DB researchers & professors
- 10th meeting: 1989, 1990, 1995, 1996, 1998, 2003, 2008, 2013, 2018, 2023, …
- 3 newest reports focus on AI/ML & Big Data:

- The Cambridge Report on Database Research, A Ailamaki, S Madden, D Abadi, G Alonso, S Amer-Yahia, M Balazinska, PA Bernstein…, April 2025, arXiv preprint arXiv:2504.11259
- Abadi, D., et al.: The Seattle Report on Database Research, Communications of the ACM, August 2022, Vol. 65, No. 8, pp. 72-79
- Abadi, D., et al.: The Beckman Report on Database Research, Communications of the ACM, February 2016, Vol. 59, No. 2, pp. 92-99

Big Data – Defining Challenge [Beckman Report 2016]

- Distribution
- Integration
- Heterogeneity
- In-memory processing
- Large-scale systems
- Data analysis
- …

Big Data – Main Characteristics



Big Data – 5 Related Challenges

- Scalable big/fast data infrastructures
- Coping with diversity in data management
- End-to-end data-to-knowledge pipeline
- Cloud services
- Role of people in data life cycle

Scalable Big/Fast Data Infrastructures

- Parallel and distributed processing
- Query processing and optimization
- New hardware
- Cost-efficient storage
- High-speed data streams
- Late-bound schemas
- Consistency
- Metrics and benchmarks

Diversity in Data Management

- No one-size-fits-all
- Cross-platform integration
- Programming models
- Data processing workflows

End-to-End Processing of Data

- Data-to-knowledge pipeline
- Tool diversity
- Tool customizability
- Open source
- Understanding data
- Knowledge bases

Cloud Services

- IaaS, PaaS, SaaS
- Elasticity (SLA)
- Data replication
- System administration and tuning
- Multitenancy (VMs)
- Data sharing
- Hybrid clouds

Roles of Humans in the Data Life Cycle

- Data producers
- Data curators (crowdsourcing)
- Data consumers
- Online communities

DB Research - New Challenges

[Beckman Report 2016] -> [Seattle Report 2022] - 1

- Correctly identified Big Data as a big theme
  – now morphed into data science, which poses big challenges
- Missed the AI/ML trend
- Promoted five directions
  – scalable data infrastructure
  – diversity in data management
  – end-to-end processing and understanding of data
  – cloud services
  – roles of humans in the data life cycle
- Made good progress, also branched out
  – e.g., into AI/ML

DB Research - New Challenges

[Beckman Report 2016] -> [Seattle Report 2022] - 2

- Beckman predicted the rise of a data-driven world
- Correctly observed that this gives us unprecedented opportunities & challenges:
  - Increasing amount and use of personal data -> data governance, ethical and fair use of data
  - Managed cloud data systems -> serverless systems, data lakes, ETL (extract, transform, load) jobs
  - Industrial Internet-of-Things (IoT)
  - Significant changes in hardware, esp. for ML/Deep Learning: FPGAs, GPUs, ASICs, …
- All of these have been true, but there are deep concerns that DBS community have failed to exploit this wealth of opportunities
  - while other communities have moved much faster.

DBS and Data Science ?

- Data Science: combines data cleaning and transformation, statistical analysis, data visualization, and ML techniques.
- Data Science [NSF CISE 2017]: “the processes and systems that enable the extraction of knowledge or insights from data in various forms, either structured or unstructured.”
- DBS technology plays a major role in Data Science: pipeline from raw input data to insights that requires use of data cleaning and transformation, data analytic techniques, and data visualization.
  - Data to insights pipeline
  - Data context and provenance
  - Data exploration at scale and data profiling
  - Declarative programming
  - Metadata management

Data Governance

- Data use policy -> GDPR, auditing
- Data privacy, e.g. differential privacy
- Ethical data science -> responsible data management

Cloud Services

- Serverless data services
- Disaggregation
- Multitenancy
- Edge and cloud
- Hybrid cloud and multi-cloud
- Auto-tuning
- SaaS cloud DB applications

Database Engines

- Heterogeneous computing
- Distributed transactions
- Data lakes
- Approximation in query answering
- Machine Learning (ML) workloads
- ML for reimaging data platform components
- Benchmarking and reproducability

DB Research - New Challenges

[Seatle Report 2022] -> [Cambridge Report 2025]

- Core Data Systems
  - Big Data everywhere, Cloud-based data systems, emerging new hardware, scalability, usability
- Human Centric Systems and Data Science
  - Big Data everywhere, data governance, NL-based querying & analysis interfaces
- ML and AI for Data Systems
  - AI/ML everywhere -> generative AI, LLMs, …
- Responsible Data Management

Core Data Systems

- Big Data everywhere
  -> usability, DB at massive scales
- Cloud-native architectures
- Disaggregated storage and compute
  -> high degree of scalability and flexibility
- Evolving hardware landscape
  -> resource-hungry AI, spec. AI accelerators

Human Centric Systems & Data Science

- Data sharing and collaboration
  -> break down data silos/lakes, enable cross-organizational analytics
- Privacy, governance, query processing across distributed datasets
- End-to-end data pipeline and workflow systems
  -> data discovery, explanations, preparation, integration and cleaning, metadata and log mgnt., versioning, analysis and visualization

ML and AI for Data Systems

- Query optimization, cost models based on ML
- Cardinality estimation, high-dimensional correlations in data distributions
- Reinforcement learning to improve physical data organization, predictive I/O
- Cloud resource management
- ML models for serverless VM management

Responsible Data Management

- Prevalence of AI models for interpreting data and making complex decisions
- Integrating data management research into responsible AI
- Decisions made during data collection and preparation impact accuracy, fairness, robustness, interpretablity, legal compliance of AI systems

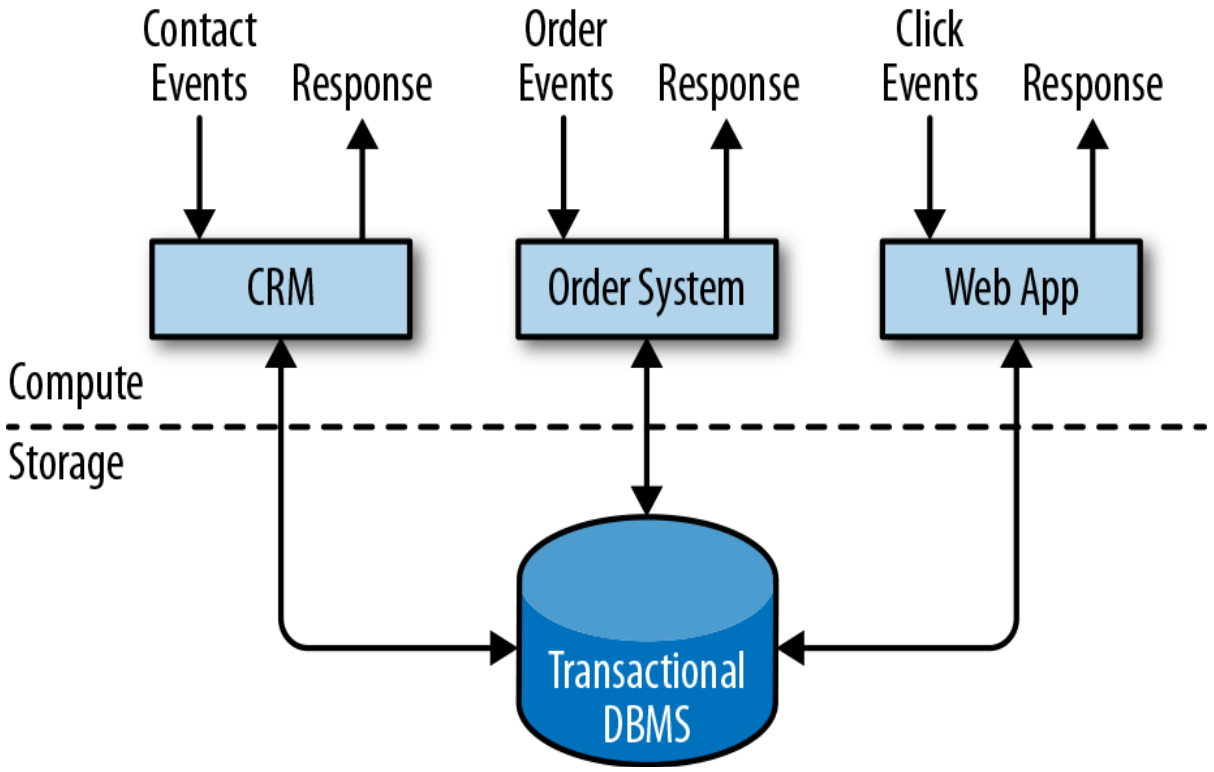
Content Overview

- Data Stream Processing (Part 1)
- Data Stream Processing (Part 2)
- Distributed Database Systems
- Heterogeneous Multi-Database Systems
- Web & XML Data Management
- Knowledge Discovery & Data Warehouses
- Machine Learning in Medicine *
- Scalable & Cloud Data Management
- Performance Analysis & Large DBS *

*Guest Lectures

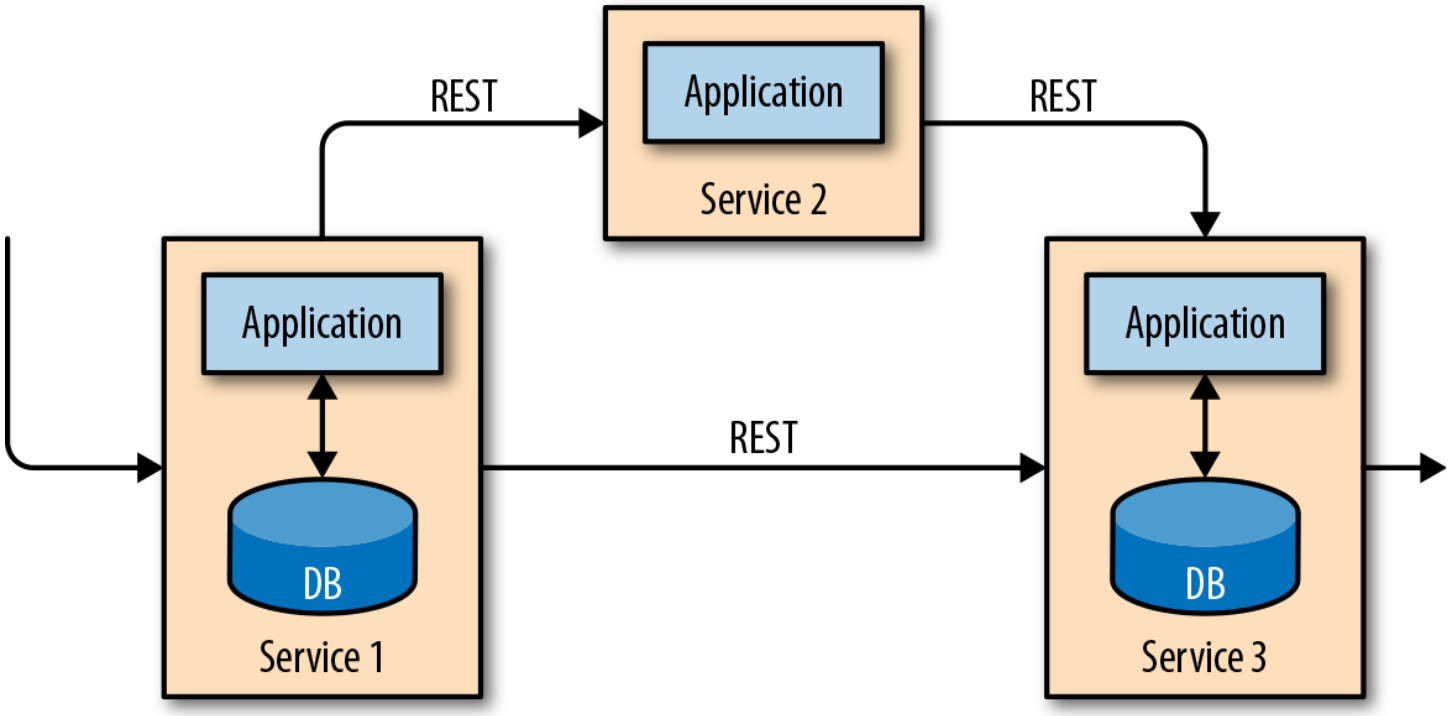
From traditional DBMS to heterogenous data processing infrastructure

Transactional Processing - Traditional



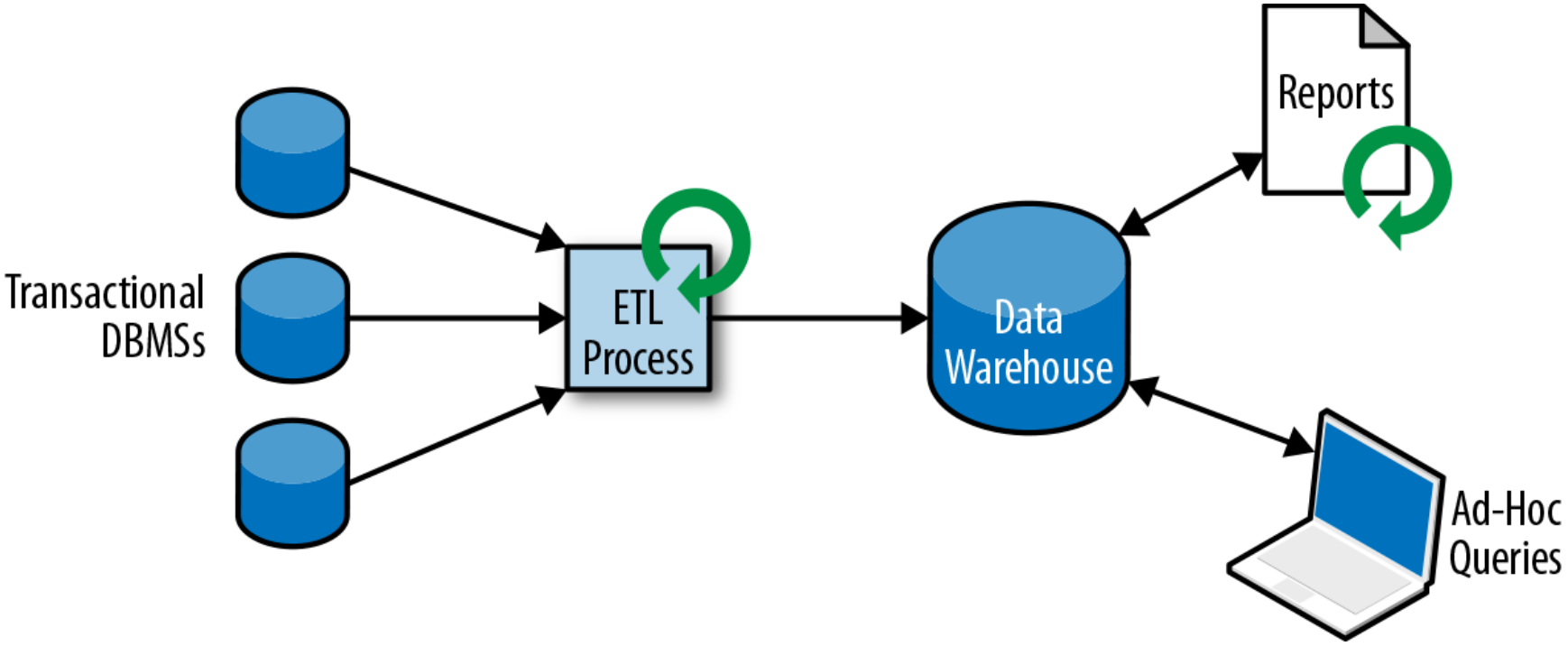
[Source: Stream Processing with Apache Flink, O´Reilly]

Transactional Processing – Micro Services



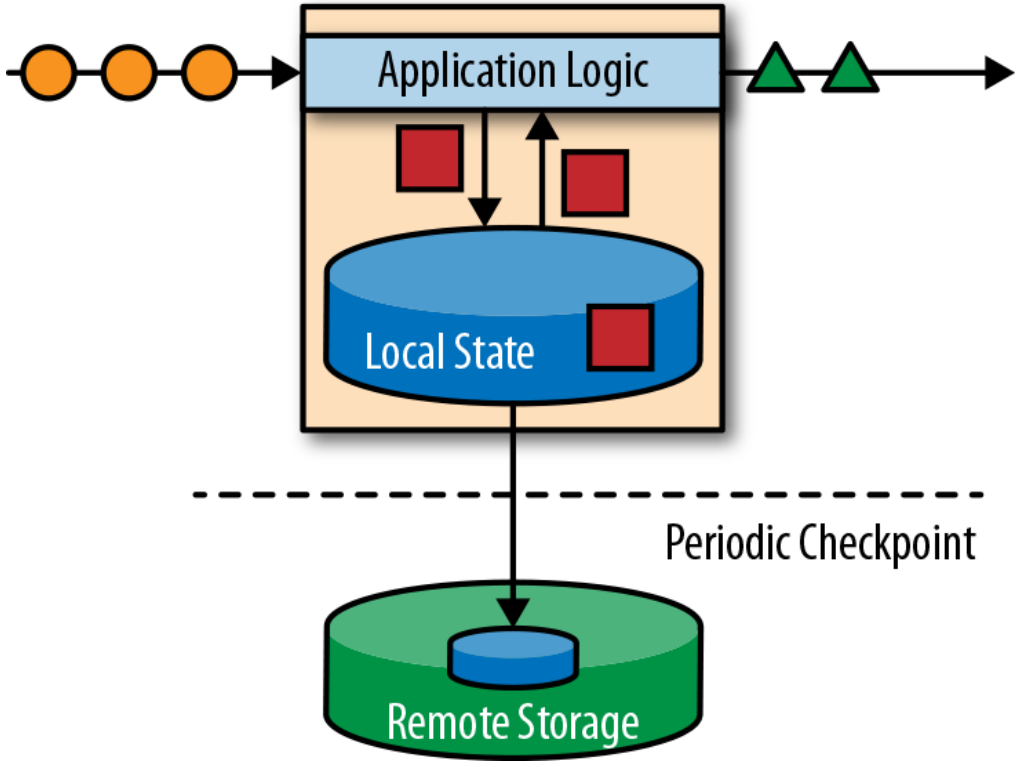
[Source: Stream Processing with Apache Flink, O´Reilly]

Analytical Processing – Data Warehouse



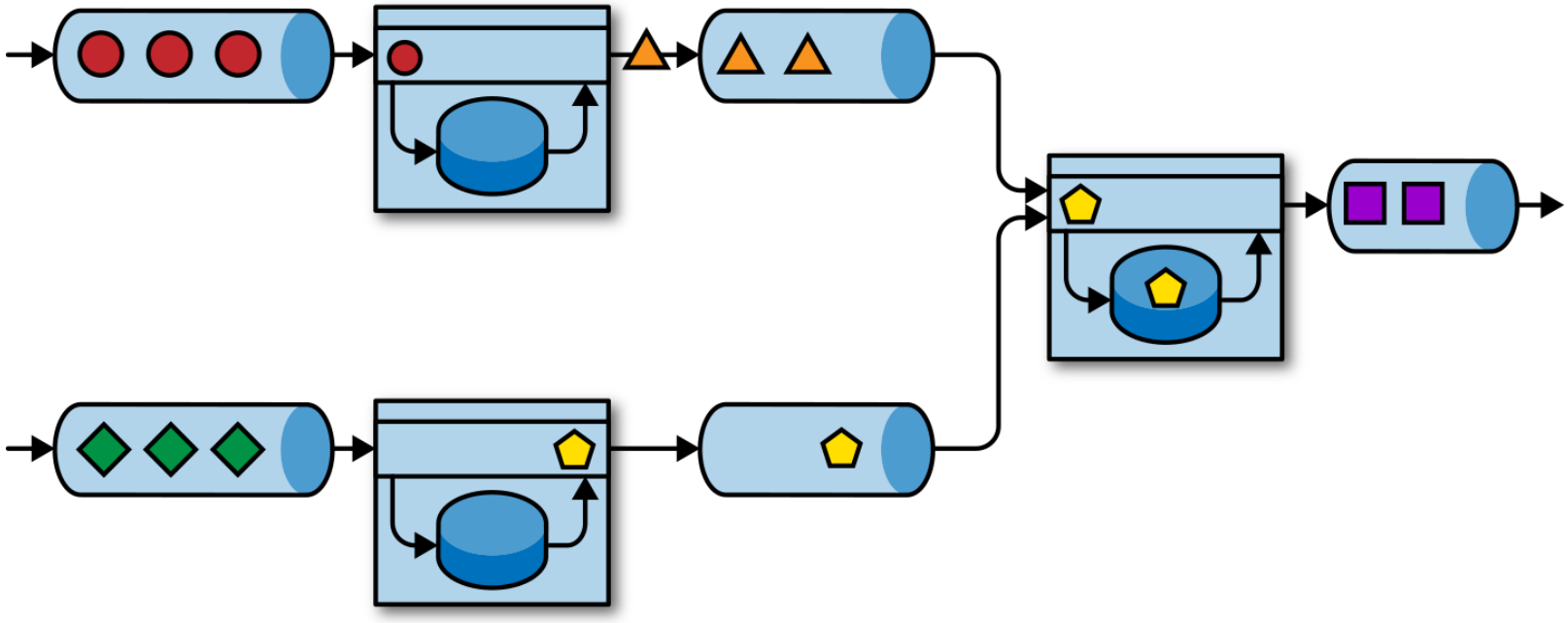
Source: Hueske, F., Kalavri, V.: Stream Processing with Apache Flink, O´Reilly, 2019

Analytical Processing – Stateful Stream Processing



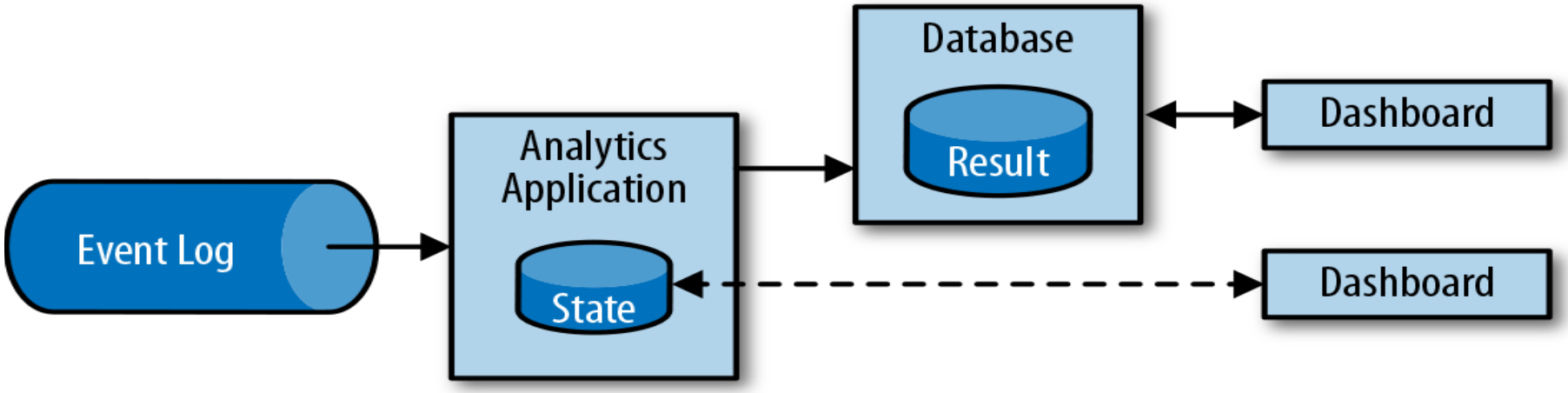
Source: Hueske, F., Kalavri, V.: Stream Processing with Apache Flink, O´Reilly, 2019

Analytical Processing – Event Driven Architecture



Source: Hueske, F., Kalavri, V.: Stream Processing with Apache Flink, O´Reilly, 2019

Streaming Analytics



Source: Hueske, F., Kalavri, V.: Stream Processing with Apache Flink, O´Reilly, 2019

Looking for Master thesis topics?

- We are part of the DKM group at Ifi:
  https://www.mn.uio.no/ifi/english/research/groups/dkm/
- Respire project:
  Responsible Explainable Machine Learning for Sleep-related Respiratory Disorders
  https://www.mn.uio.no/ifi/english/research/projects/respire/index.html
- Parrot project:
  Privacy Engineering for Real-Time Analytics in Human-Centered Internet-of-Things
  https://www.mn.uio.no/ifi/english/research/projects/parrot/index.html

- Contact: plageman@ifi.uio.no

Data(base) Integration
Multi-Database Systems (MDBS)

Vera Goebel
Department of Informatics, University of Oslo

Interoperability Problem

- Interoperability at the application level
  Data Integration from various data sources including non-database sources ->
  Data Stream Processing Systems, Data Lakes
- Interoperability at the database level
  Bottom-up design process ->
  Multi-Database Systems, Data Warehouses

- Data exchange → Web Data Management

Integration Alternatives

(complementary) for different applications/usage

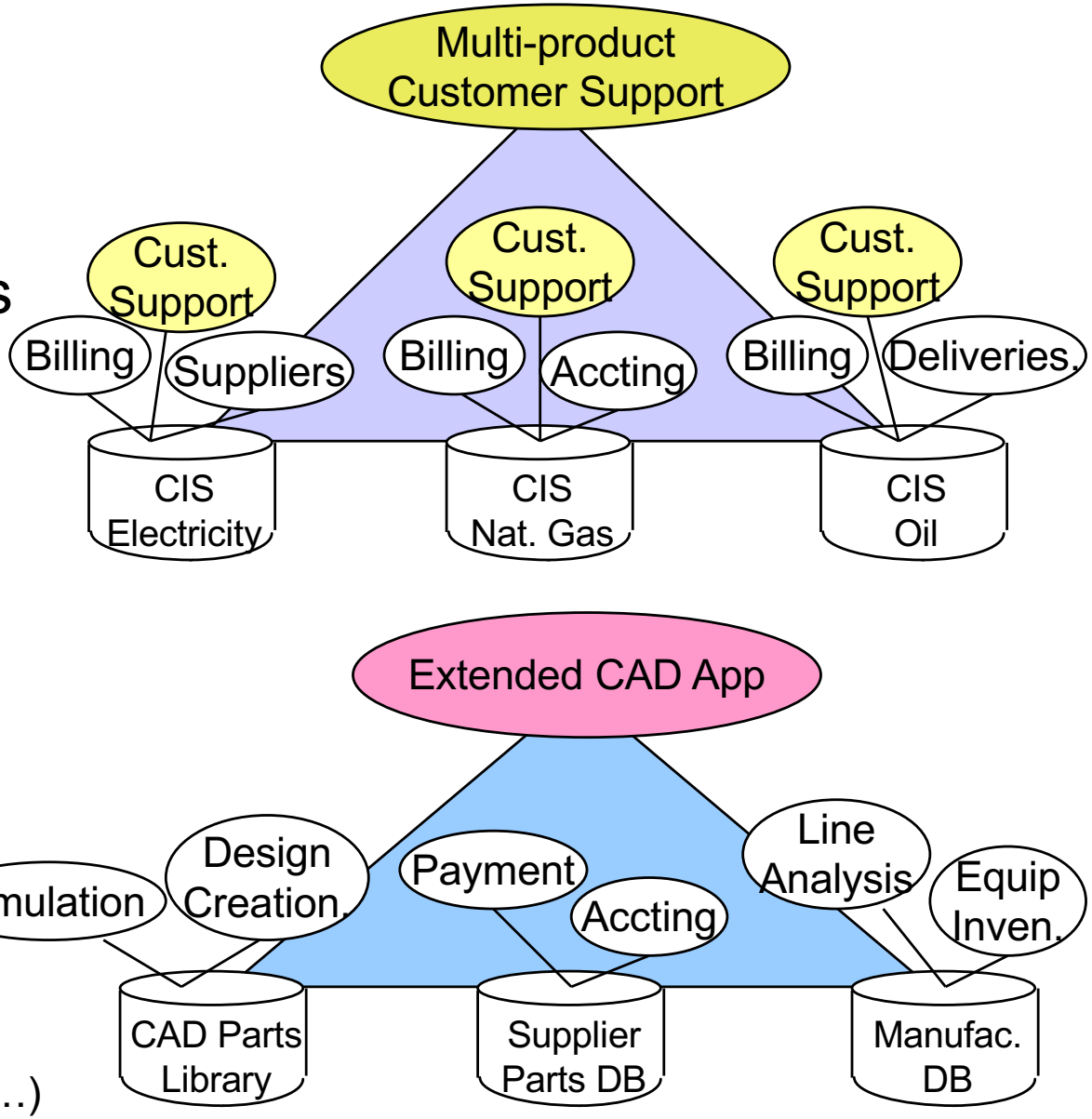
- Physical integration (OLAP) ->
  Data Warehouses, Data Lakes
  - Source databases integrated and integrated database is materialized (ETL - extract-transform-load)
  - Multi-dimensional DB for complex data analysis (ML)
- Logical integration (OLTP) ->
  Multi-DBS (MDBS)
  - Global conceptual schema is virtual and not materialized
  - Data control and availability, multi-user, high throughput

Heterogeneous / Federated / Multi-Database Systems (MDBS)

- Why Heterogeneous MDBS?
- Applications
- Architectures for MDBS
- Main Problems:
  - Global Data Model
  - Query Processing
  - Query Optimization?
  - Transaction Management?

Applications

- Multitude of extensive, isolated data agglomerations managed by different DBS
  - Similar data
    - Ex: 3 Customer Info Systems
  - Dissimiliar data
    - Ex: Extended CAD Application
- Extension of data and management software because of new and/or extended applications
- Heterogeneous application domains (e.g., CIM, CAD, Biz-mgmt, …)



Requirements for MDBS

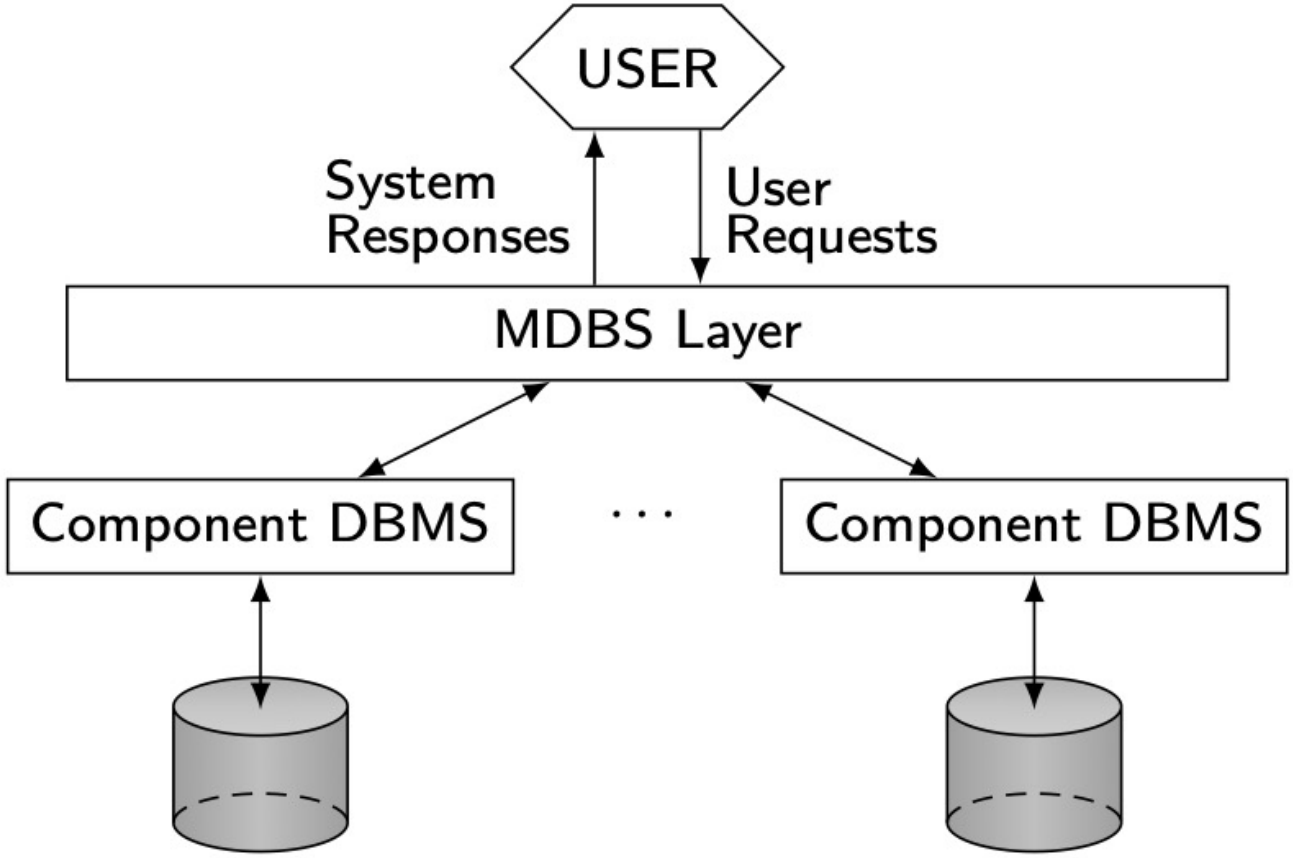
Integration of Heterogeneous DBSs

-> queries across MDBS (combine heterogeneous data)
-> heterogeneous information structures
-> avoid redundancy
-> access (query) language transparency

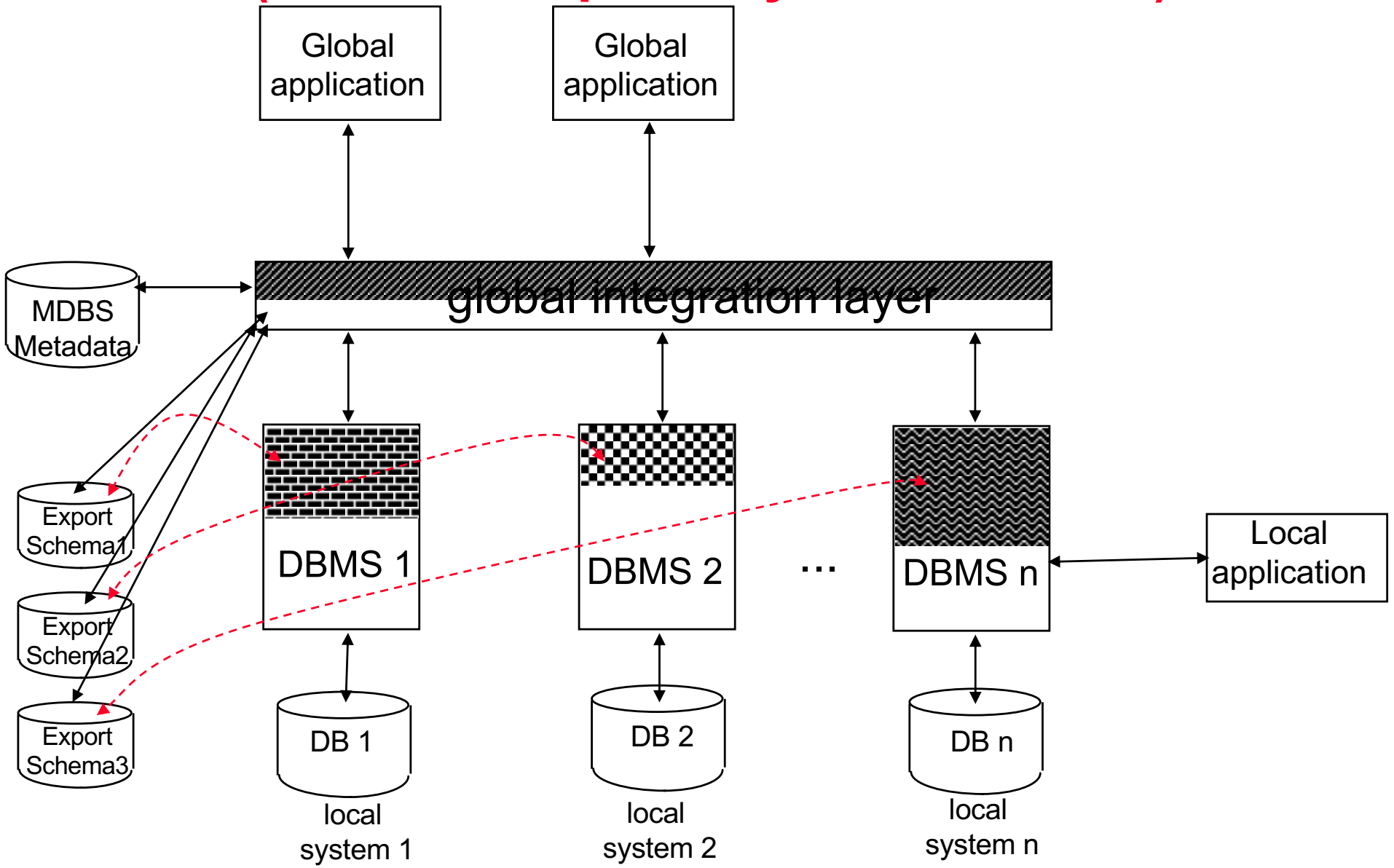
- **"Open" system**
  support for integration of existing data models and DBSs, as well as their schemata and databases

- **Constraints**
  -> retain autonomy of DBSs to be integrated
  -> avoid modifications of existing local applications
  -> define a viable global data model for global applications

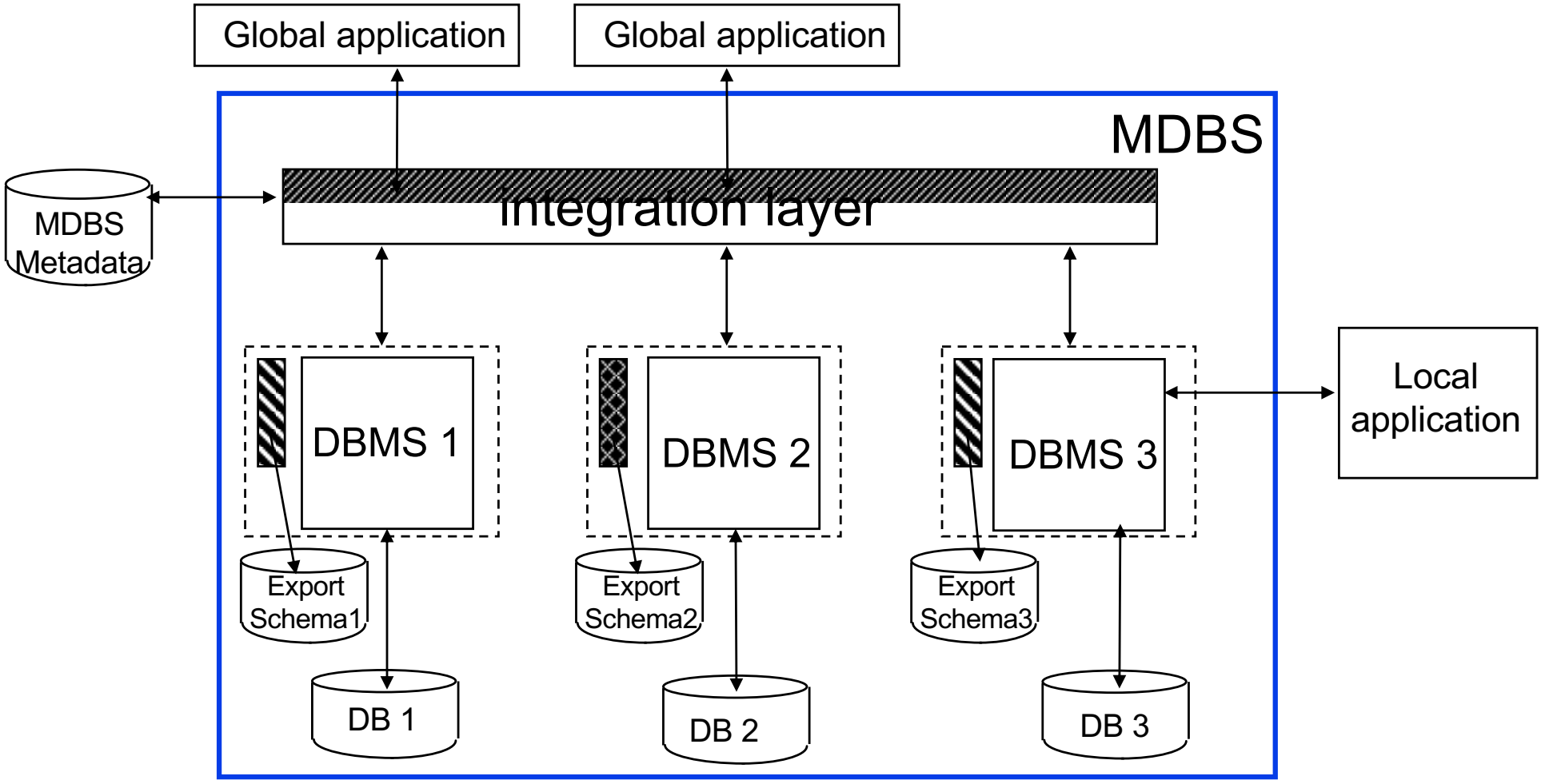
Database Integration – Multi-DBS Architecture



MDBS (federation, partially autonomous)



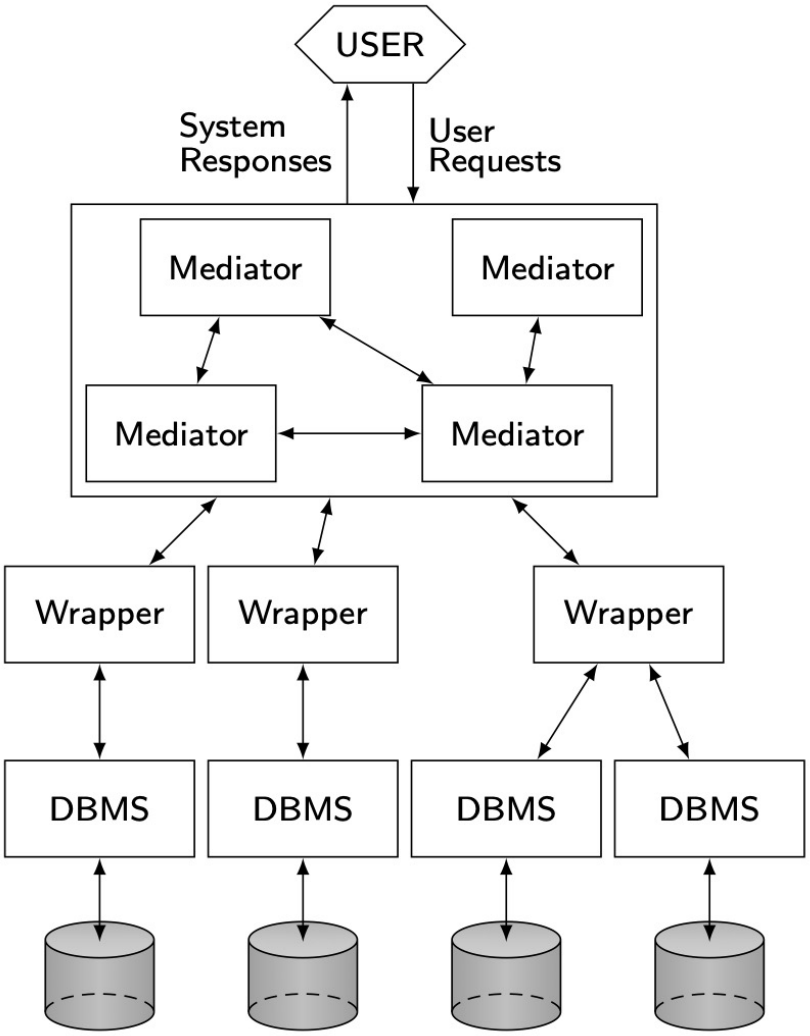
MDBS (fully autonomous)



MDBS Server or MDBS Proxy

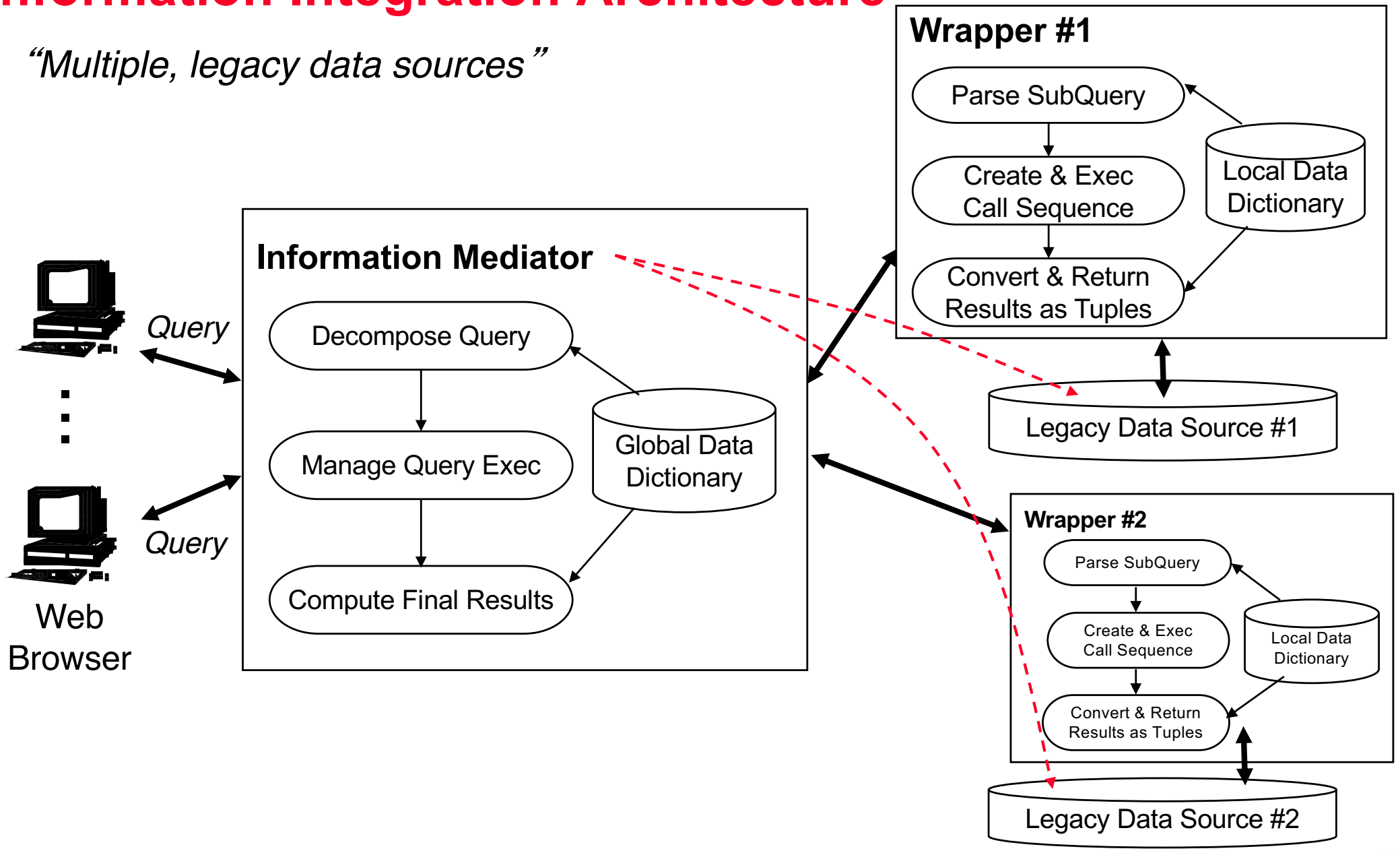
- Runs on the local DB site
- Typically includes some code that is specific to the local DB type

Mediator/Wrapper Architecture



Information Integration Architecture

"Multiple, legacy data sources"



Integration Layer

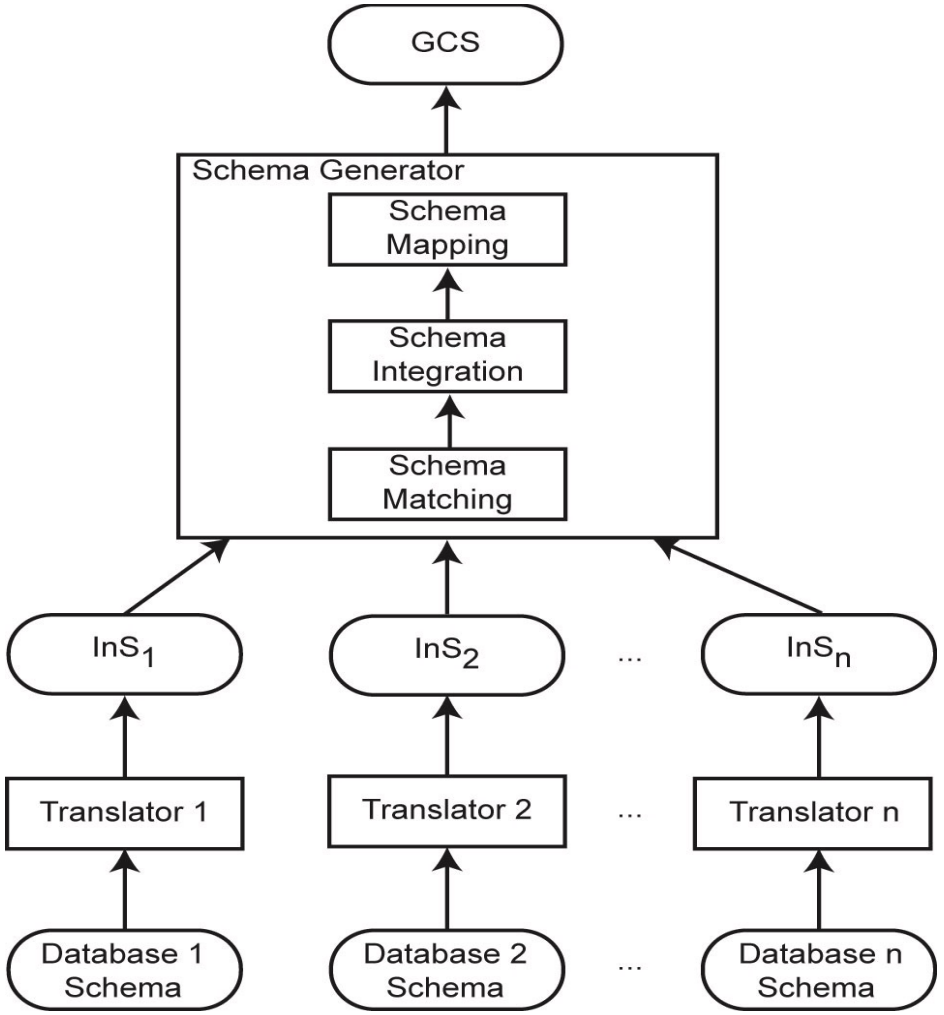
- **Global data model**:
  - Local data models: any kind of data model possible, e.g., object-oriented, relational, entity-relationship, hierarchical, network-oriented, ...
  - Global data model: must comprise modeling concepts and mechanisms to express the features of the local data models
- Global schema and metadata management
- Distributed query processing and optimization

- Primarily read-only queries/transactions at the global level
  - Autonomous local DBS can run all kinds of queries/transactions -> unknown at the global integration layer/level

Database Integration Process

- Schema translation: Component database schemas translated to a common intermediate canonical representation
- Schema generation: Intermediate schemas are used to create a global conceptual schema
  - Schema Matching
  - Schema Integration
  - Schema Mapping
- Query rewriting and processing
- Query optimization

Database Integration Process



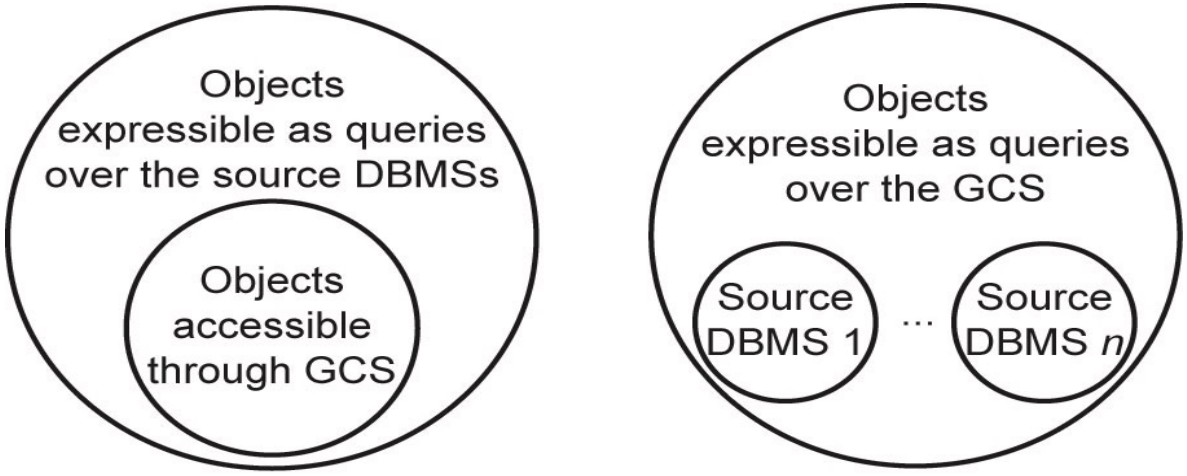
Bottom-up Design

- Problem: Given existing databases with their Local Conceptual Schemas (LCSs), how to integrate the LCSs into a Global Conceptual Schema (GCS)

- GCS (mediated schema) is defined first
  - Map LCSs to this schema
  - As in data warehouses
- GCS is defined as an integration of parts of LCSs
  - Generate GCS and map LCSs to this GCS

GCS/LCS Relationship

- Local-as-view
  - The GCS definition is assumed to exist, and each LCS is treated as a view definition over it
- Global-as-view
  - The GCS is defined as a set of views over the LCSs



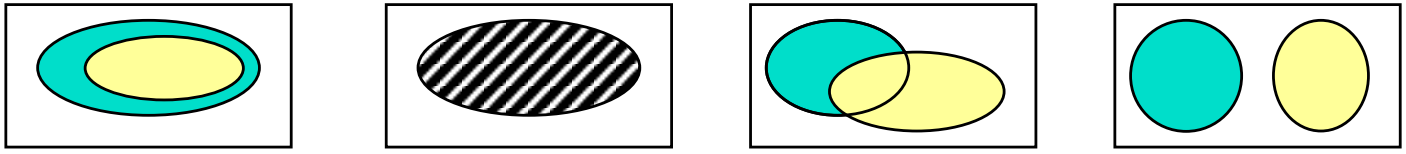
(a) GAV (b) LAV

Schema Generation

- Schema matching
  - Finding the correspondences between multiple schemas
- Schema integration
  - Creation of the GCS (or mediated schema) using the correspondences
- Schema mapping
  - How to map data from local databases to the GCS
- Important: sometimes the GCS is defined first and schema matching and schema mapping is done against this target GCS

Schema Homogenization

- Schema Translation
  - Map each local schema to the language of the global data model
    - Ex: a Relational schema to an Object-oriented schema
- Schema Integration
  - For *N* translated, local schemas
    - Pairwise integration, X-at-a-time integration, One-step integration
  - Determine ”common semantics” of the schemas



  - Make the ”same things” be ”one thing” in the integrated schema
  - Resolve conflicts
    - structural and semantic

Adequate design tools are not available

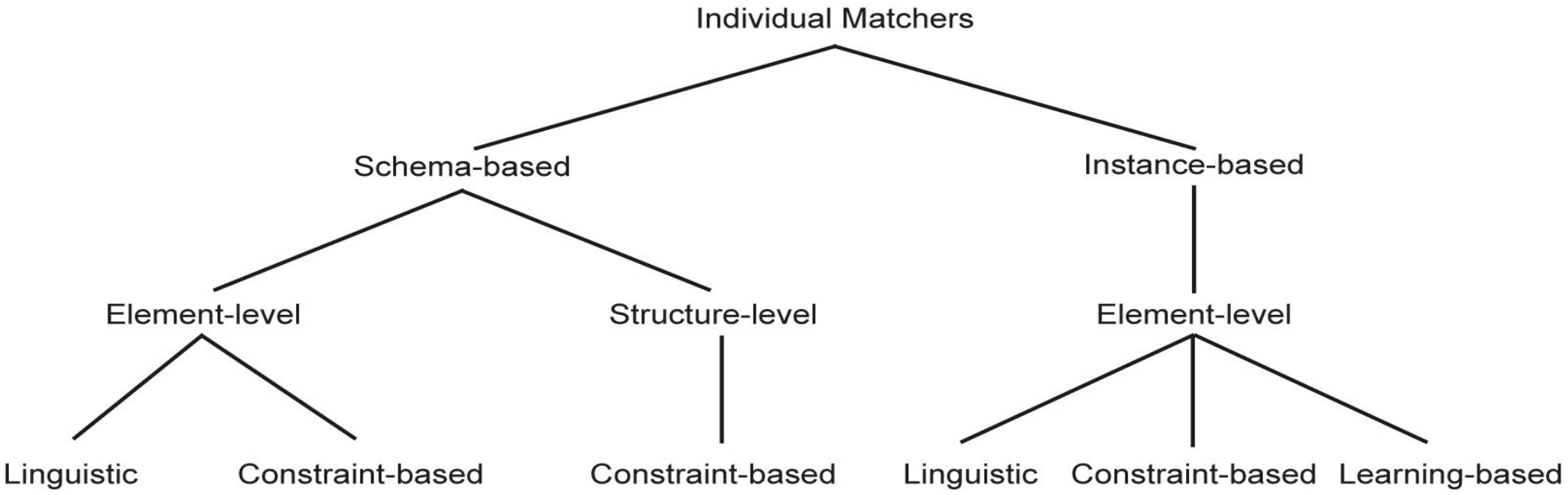
Schema Matching

- Schema heterogeneity
  - Structural heterogeneity
    - Type conflicts
    - Dependency conflicts
    - Key conflicts
    - Behavioral conflicts
  - Semantic heterogeneity
    - More important and harder to deal with
    - Synonyms, homonyms, hypernyms
    - Different ontology
    - Imprecise wording

Schema Matching (cont.)

- Other complications
  - Insufficient schema and instance information
  - Unavailability of schema documentation
  - Subjectivity of matching
- Issues that affect schema matching
  - Schema versus instance matching
  - Element versus structure level matching
  - Matching cardinality

Schema Matching Approaches



Schema Conflicts

- Name
  - Different names for equivalent entities, attributes, relationships, etc.
  - Same name for different entities, attributes, …
- Structure
  - Missing attributes
  - Missing but implicit attributes
- Relationship
  - One-to-many, many-to-many
- Entity versus Attribute (inclusion)
  - One attribute or several attributes
- Behavior
  - Different integrity constraints

Data Representation Conflicts

- Different representation for equivalent data
  - Different units
    - Celsius ↔ Farenheit; Kilograms ↔ Pounds; Liters ↔ Gallons;
  - Different levels of precision
    - 4 decimal digits versus 2 decimal digits
    - Floating point versus integer
  - Different expression denoting same information
    - Enumerated Value sets that are not one-to-one
      - {good, ok, bad} versus {one, two, three, four, five}

Conflict Resolution

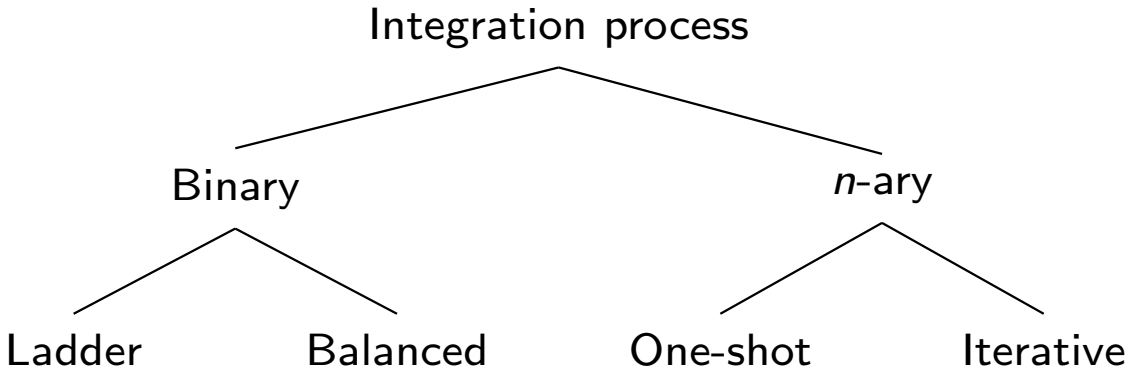
- Renaming entities and attributes
  - Pick one name for the same things
  - Use unique prefixes for different things
- Homogenizing representations
  - Use conversions and mappings
    - stored programs in relational systems
    - methods in OO systems
    - auxiliary schemas to store conversion rules/code
- Homogenizing attributes
  - Use type coercion (e.g., integer to float)
  - Attribute concatenation (e.g., first name || last name)
  - For missing attributes, assign default values
- Homogenizing an attribute and an entity
  - Extract an attribute from the entity
  - Create an entity from the attribute

Conflict Resolution

- Horizontal joins
  - Union compatible
    - For missing attributes, assign default values or compute implicit values
  - Extended union compatible
    - Use generalization
      - Define a virtual class containing common attributes
    - Subclasses of the generalization
      - Provide specialized values and compute attribute values for generalized attributes
    - See earlier example
      - class Person generalizes
        class Student and class Employee
- Vertical joins
  - Many and many to one
- Mixed Joins
  - Vertical and horizontal joins in combination

Schema Integration

- Use the correspondences to create a GCS
- Mainly a manual process, rules can help



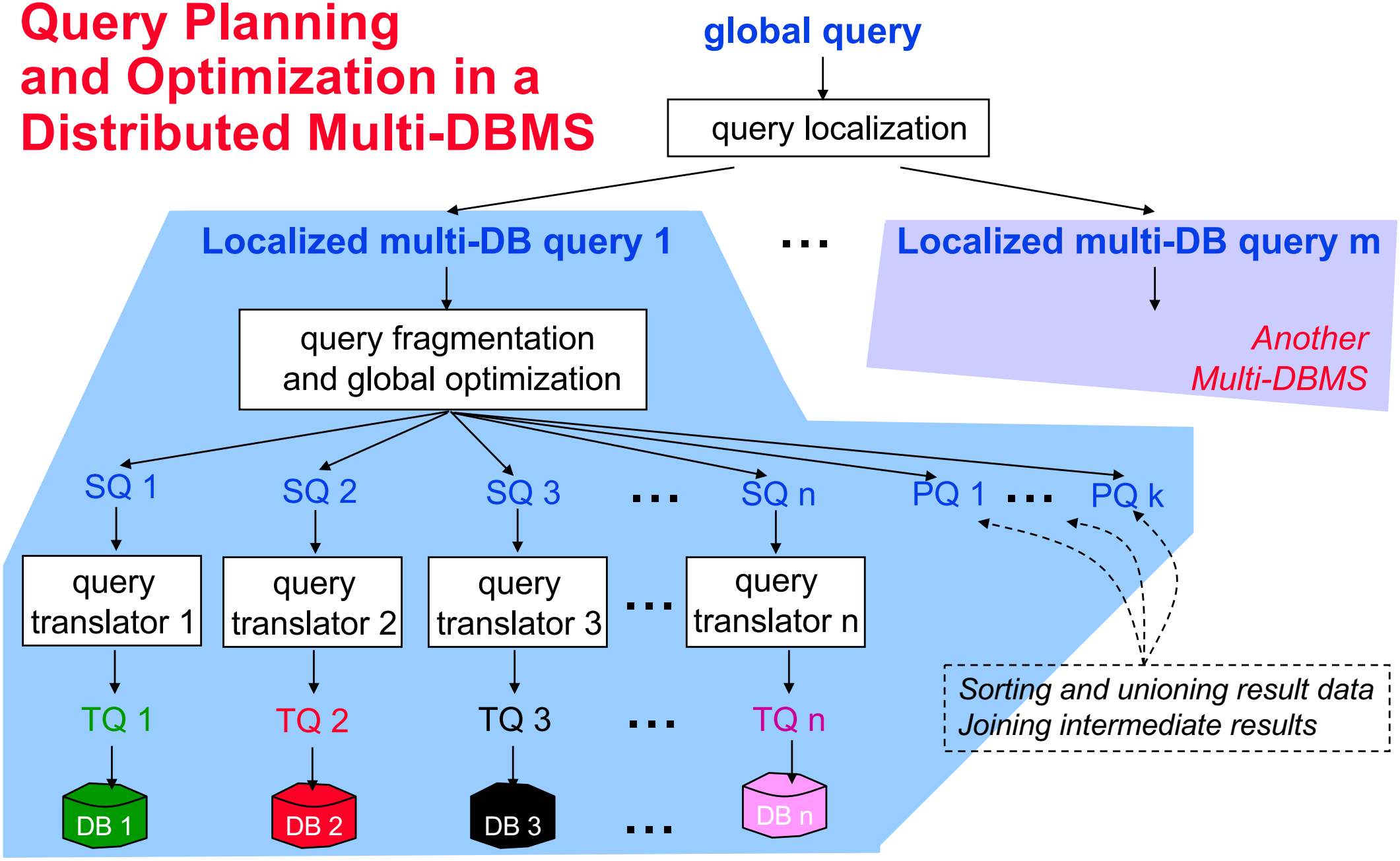
Schema Mapping

- Mapping data from each local database (source) to GCS (target) while preserving semantic consistency as defined in both source and target.
- Data warehouses ⇒ actual translation
- Data integration systems ⇒ discover mappings that can be used in the query processing phase
- Mapping creation
- Mapping maintenance

Global Schema Management

- Global schema = sum of all local exported schemata
- Global schema definition facilities provide mechanisms for handling the full spectrum of schematic differences that may exist among the heterogeneous local schemata.
- Data is stored in the local component DBS.
- Global dictionary information is used to query and manipulate the data. The
- Global queries are translated into equivalent queries of the local languages supported by the local DBS

Query Planning and Optimization in a Distributed Multi-DBMS



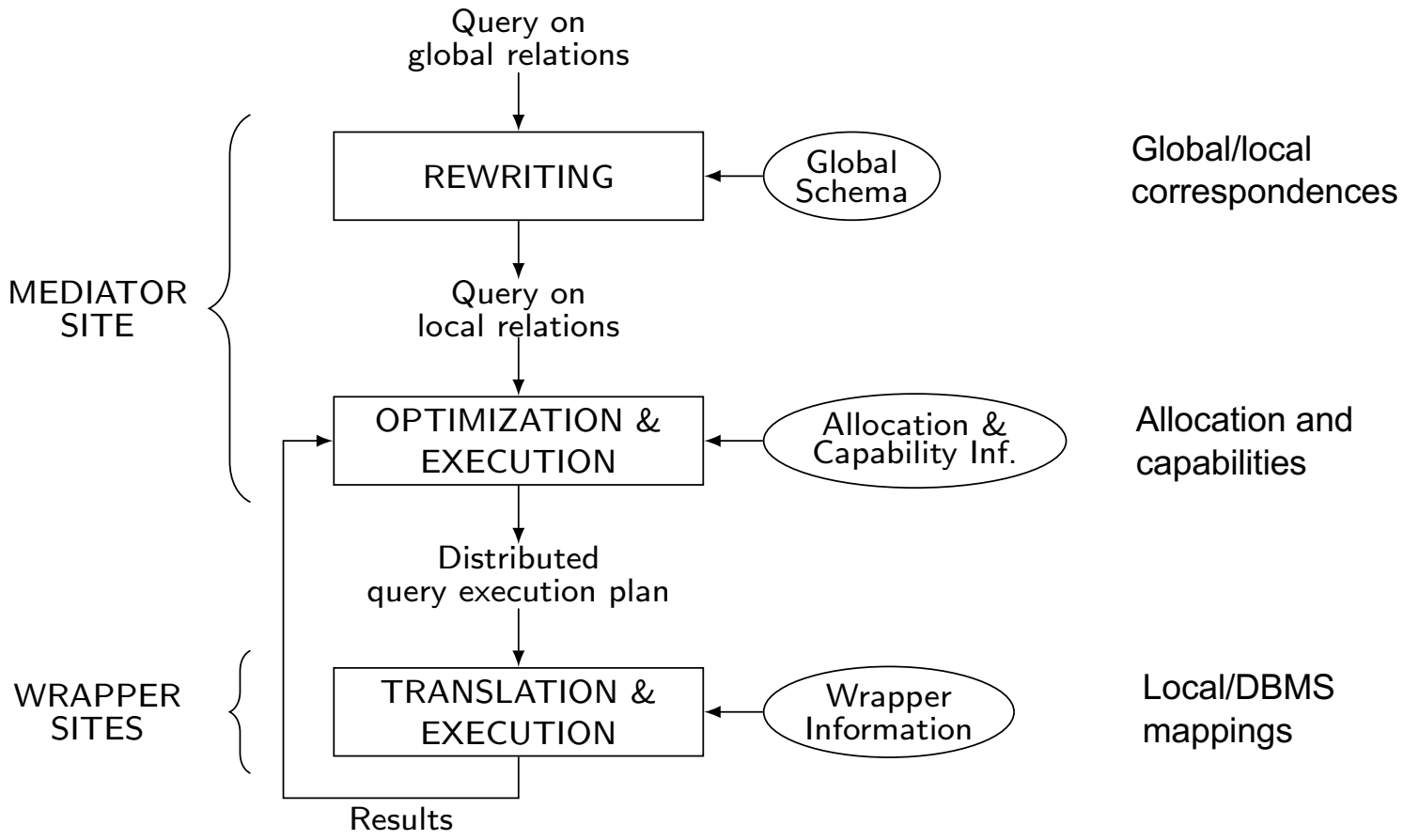
Issues in MDBS Query Processing

- Component DBMSs are autonomous and may range from full-fledge relational DBMS to flat file systems
  - Different computing capabilities
    - Prevents uniform treatment of queries across DBMSs
  - Different processing cost and optimization capabilities
    - Makes cost modeling difficult
  - Different data models and query languages
    - Makes query translation and result integration difficult
  - Different runtime performance and unpredictable behavior
    - Makes query execution difficult

Component DBMS Autonomy

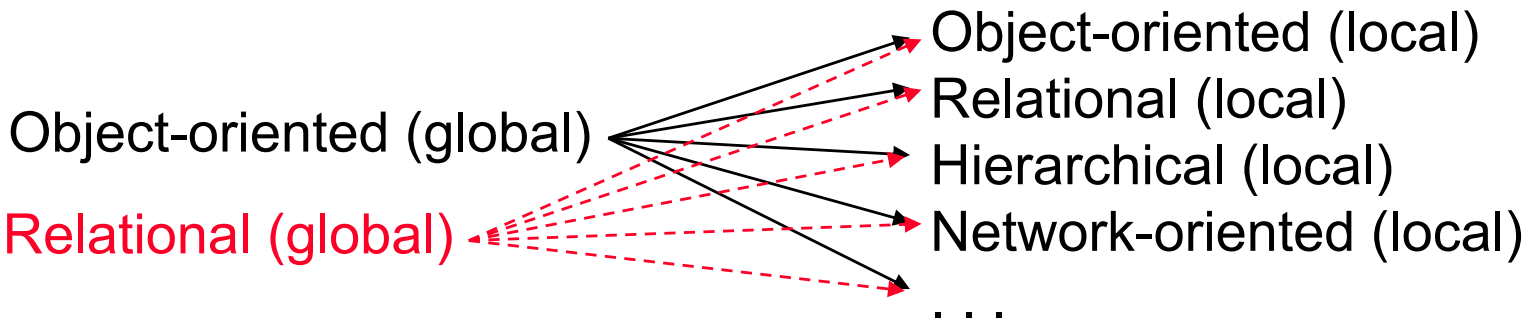
- Communication autonomy
  - The ability to terminate services at any time
  - How to answer queries completely?
- Design autonomy
  - The ability to restrict the availability and accuracy of information needed for query optimization
  - How to obtain cost information?
- Execution autonomy
  - The ability to execute queries in unpredictable ways
  - How to adapt to this?

MDBS Query Processing Steps



Query Translation

When a query language of a local DBS is different from the global query language, each export schema subquery for the local DB needs to be translated from the global language to the target language.



Weaker target languages do not support the same operations, emulate required operations in post-processing

- retrieve more data than requested by the query
- then post-process data to compute correct query result

Query Fragmentation

- Similar to query fragmentation problem for homogeneous distributed DBSs
- Complicating factors:
  - Autonomy
    - Little information about "how" the subquery will be executed by the Local DBS
  - Heterogeneous Data Definition Languages
    - Weaker modeling languages do not support the same manipulation "features"
    - Must use multiple techniques in order to define a consistent global data model
    - Query fragmentation must produce a set of subqueries that reverse the operations used to create/define the global schema
- Processing Steps:

(1) Replace names from the global schema with "fullnames" from the export schemas

(2) If a subquery involves multiple export schemas, then break the query into queries that operate on one export schema and insert data communication operators to exchange intermediate results between local database systems

Global Query Optimization

- Similar to global query optimization for homogeneous distributed DBSs (many algorithms can be used directly)
- But only possible under the following assumptions:
  - No data inconsistency (the global schema correctly represents the semantics of disjoint, overlapping, and conflicting data)
  - Know the characteristics of local DBSs
    - e.g., statistical info on data cardinalities and selectivities are available
  - Can transfer partial data results between different local DBSs
    - Major impact on post-processing plans
- Primary Considerations:
  - Post-processing Strategy
  - Parallel Execution Possibilities
  - Global Cost Function/Estimation

Post-Processing Strategies

- Three Strategies:

1) Control site performs all intermediate and post-processing operations (I&PP-ops)
   - Heavy workload; minimal parallelism

2) Control site performs I&PP-ops for multi-DB results; Multi-DB managers, and HDBMS agents on the local database sites perform I&PP-ops for DBSs within one multi-DB environment
   - Better work load balance; more parallelism

3) Use strategy #2 and use “pushdown” to get the local database systems to perform I&PP-ops
   - Possible if local DBMS can read intermediate results from external sources, and sort, join, etc. can be directly invoked

Global Cost Estimation

- Differs from cost estimation in homogeneous distributed DBSs
  - Little (or no) info on QP algorithms and data statistics in local DBS
- Cost Estimation Function
  - Cost to execute each subquery on the local DBMSs
  - Cost to execute all I&PP-ops
    - via pushdown or by any HDBMS agent/service
- Use a simplified cost function

Cost = Initialization cost
+ cost to retrieve a set of objects
+ cost to process a set of objects

- Run test queries on the local DBSs to get time estimates for ops
  - Selection, with and without an index
  - Join (testing for different algorithms: sort, hash, or indexed based algorithms)

Query Optimization and Execution

- Takes a query expressed on local relations and produces a distributed QEP to be executed by the wrappers and mediator
- Three main problems
  - Heterogeneous cost modeling
    - To produce a global cost model from component DBMS
  - Heterogeneous query optimization
    - To deal with different query computing capabilities
  - Adaptive query processing
    - To deal with strong variations in the execution environment

Heterogeneous Cost Modeling

- Goal: determine the cost of executing the subqueries at component DBMS
- Three approaches
  - Black-box: treats each component DBMS as a black-box and determines costs by running test queries
  - Customized: customizes an initial cost model
  - Dynamic: monitors the run-time behavior of the component DBMS and dynamically collect cost information

Query Translation and Execution

- Performed by wrappers using the component DBMS
  - Conversion between common interface of mediator and DBMS-dependent interface
    - Query translation from wrapper to DBMS
    - Result format translation from DBMS to wrapper
  - Wrapper has the local schema exported to the mediator (in common interface) and the mapping to the DBMS schema
  - Common interface can be query-based (e.g. ODBC or SQL/MED) or operator-based
- In addition, wrappers can implement operators not supported by the component DBMS, e.g. join

Wrapper Management Issues

- Wrappers mostly used for read-only queries
  - Makes query translation and wrapper construction easy
  - DBMS vendors provide standard wrappers
    - ODBC, JDBC, ADO, etc.
- Updating makes wrapper construction harder
  - Problem: heterogeneity of integrity constraints
    - Implicit in some legacy DB
  - Solution: reverse engineering of legacy DB to identify implicit constraints and translate in validation code in the wrapper
- Wrapper maintenance
  - schema mappings can become invalid as a result of changes in component DB schemas
    - Use detection and correction, using mapping maintenance techniques

Transaction Management?

- Local transactions: access data at a single site outside of the global MDBS control.
- Global transactions: are executed under the MDBS control
  -> mostly read-only

Local DBMSs have three types of autonomy:

Autonomy Type	Definition	Resulting Problem
Design	No changes can be made to the local DBMS software to support the HDBMS	Non-serializable schedule for global transactions
Execution	Each local DBMS controls execution of global subtransactions and local transactions (the commit/abort decision)	Non-atomic & non-durable global transactions
Communication	Local DBMS do not communicate with each other and they do not exchange execution control information	Distributed deadlock can not be detected

Distributed Database Systems

Vera Goebel

Department of Informatics

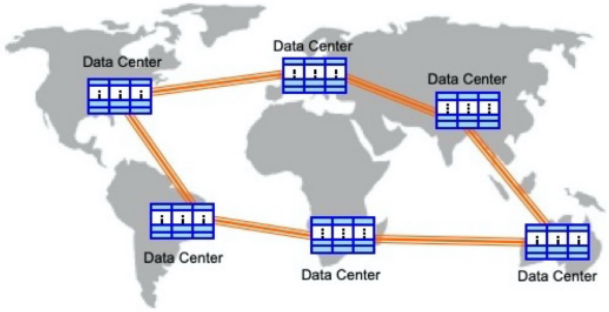
University of Oslo

Outline:
- Terminology and definitions
- Architectures
- Problems, concepts and approaches

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



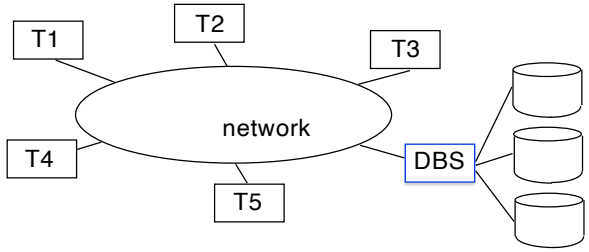
What is a Distributed Database System?

Distributed DB: collection of multiple, logically interrelated databases distributed over computer network.

Distributed DBMS: software managing distributed DB, provides access mechanism to make distribution transparent to users.

Centralized DBS

- logically integrated
- physically centralized



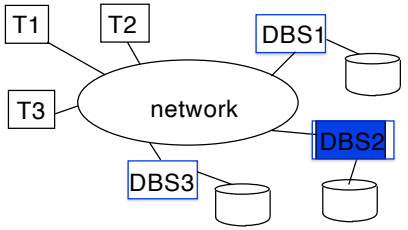
Traditionally: one large mainframe DBMS + n "stupid" terminals

5

5

Distributed DBS (P2P Dist. DBS)

- Data logically integrated (i.e., access based on one schema)
- Data physically distributed among multiple database nodes
- Processing is distributed among multiple database nodes



Why a Distributed DBS?

Performance via parallel execution
- more users
- quick response

More data volume
- max disks on multiple nodes

Traditionally: m mainframes for the DBMSs + n terminals

6

6

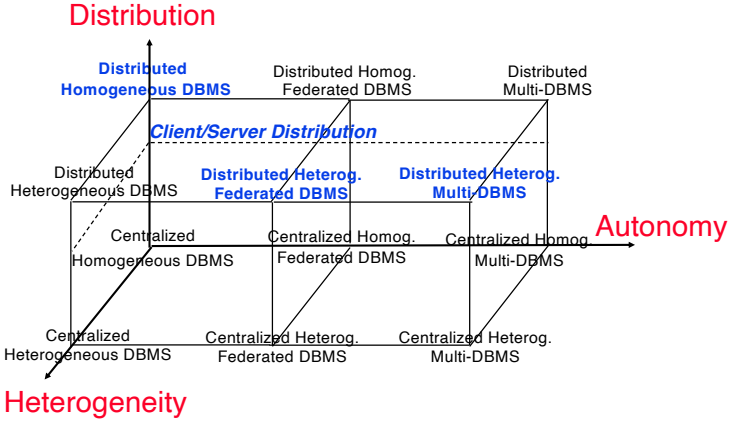
Distributed DBMS

- Advantages:
  - Improved performance
  - Efficiency
  - Extensibility (addition of new nodes)
  - Transparency of distribution
    - Storage of data
    - Query execution
  - Autonomy of individual nodes
- Problems:
  - Complexity of design and implementation
  - Data consistency
  - Safety
  - Failure recovery

7

7

Classification of Distributed DBMS Alternatives



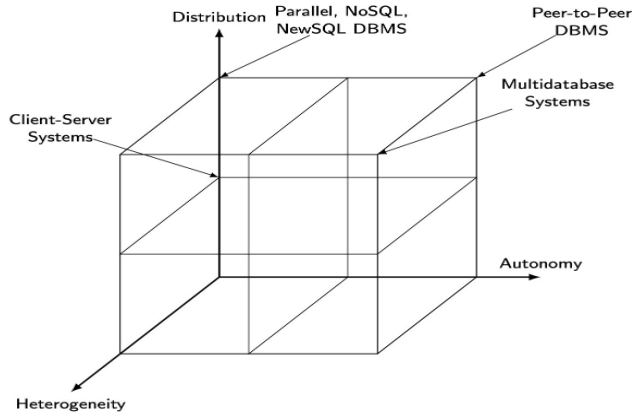
8

8

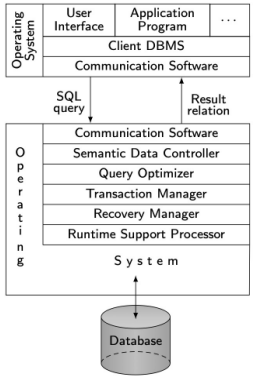
Dimensions of the Problem

- Distribution
  - Whether the 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
  - Not well understood and most troublesome
  - 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.

Modern Distributed DBMS Alternatives



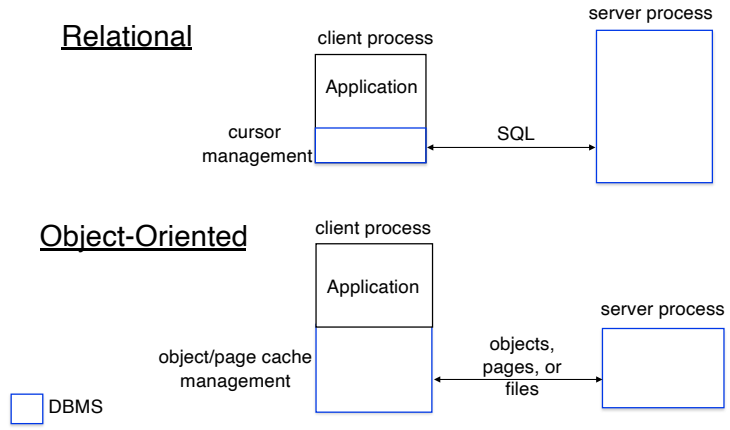
Client/Server Architecture



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

Client/Server Architectures



13

13

Comparison of Client/Server Architectures

Page & File Server

- Simple server design
- Complex client design
- Fine grained concurrency control difficult
- Very sensitive to client buffer pool size and clustering

Object Server

- Complex server design
- "Relatively" simple client design
- Fine-grained concurrency control
- Reduces data movement, relatively insensitive to clustering
- Sensitive to client buffer pool size

Conclusions:

- No clear winner
- Depends on object size and application's object access pattern
- File server ruled out

14

14

Cache Consistency in Client/Server Architectures

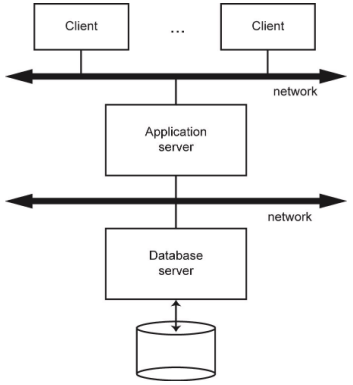
	Synchronous	Asynchronous	Deferred
Avoidance-Based Algorithms	Client sends 1 msg per lock to server; Client waits; Server replies with ACK or NACK.	Client sends 1 msg per lock to the server; Client continues; Server invalidates cached copies at other clients.	Client sends all write lock requests to the server at commit time; Client waits; Server replies when all cached copies are freed.
Detection-Based Algorithms	Client sends object status query to server for each access; Client waits; Server replies.	Client sends 1 msg per lock to the server; Client continues; After commit, the server sends updates to all cached copies.	Client sends all write lock requests to the server at commit time; Client waits; Server replies based on W-W conflicts only.

Best Performing Algorithms

15

15

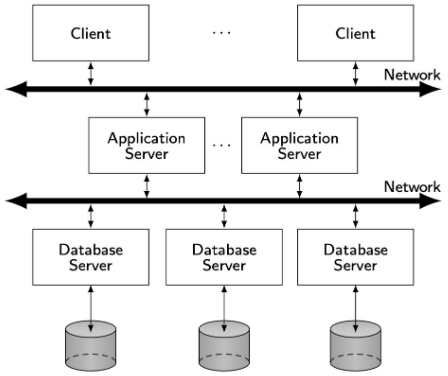
Database Server



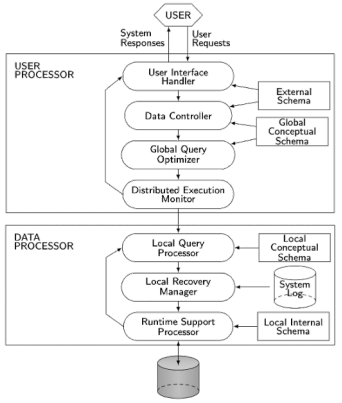
16

16

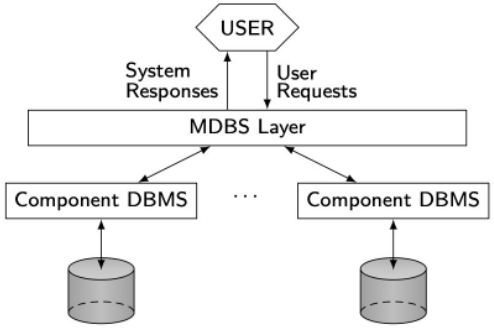
Distributed Database Servers



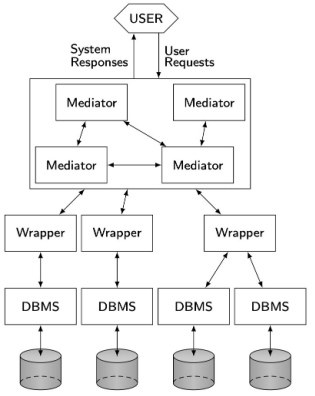
Peer-to-Peer (P2P) Component Architecture



Multi-DBS Components & Execution

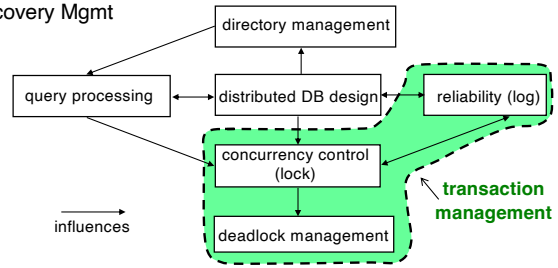


Mediator/Wrapper Architecture



Distributed DBMS Functionality

Distributed Database Design
Distributed Directory/Cataloge Mgmt
Distributed Query Processing and Optimization
Distributed Transaction Mgmt
- Distributed Concurreny Control
- Distributed Deadlock Mgmt
- Distributed Recovery Mgmt



21

21

Distributed Database Design

- horizontal fragmentation: distribution of "rows", selection
- vertical fragmentation: distribution of "columns", projection
- hybrid fragmentation: "projected columns" from "selected rows"
- allocation: which fragment is assigned to which node?
- replication: multiple copies at different nodes, how many copies?

- Design factors:
  - Most frequent query access patterns
  - Available distributed query processing algorithms
- Evaluation Criteria
  - Cost metrics for: network traffic, query processing, transaction mgmt
  - A system-wide goal: Maximize throughput or minimize latency

22

22

Distributed Directory and Catalogue Management

- Directory Information:
  - Description and location of records/objects
    - Size, special data properties (e.g.,executable, DB type, user-defined type, etc.)
    - Fragmentation scheme
  - Definitions for views, integrity constraints
- Options for organizing the directory:
  - Centralized ← Issues: bottleneck, unreliable
  - Fully replicated ← Issues: consistency, storage overhead
  - Partitioned ← Issues: complicated access protocol, consistency
  - Combination of partitioned and replicated ← e.g., zoned, replicated zoned

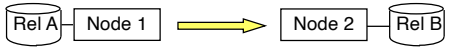
23

23

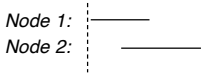
Distributed Query Processing and Optimization

- Construction and execution of query plans, query optimization
- Goals: maximize parallelism (response time optimization)
  minimize network data transfer (throughput optimization)
- Basic Approaches to distributed query processing:

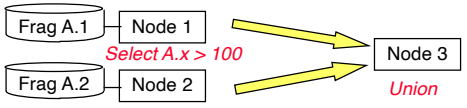
Pipelining – *functional decomposition*



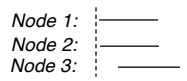
Processing Timelines



Parallelism – *data decomposition*



Processing Timelines



24

24

Creating the Distributed Query Processing Plan

- Factors to be considered:
  - distribution of data
  - communication costs
  - lack of sufficient locally available information

- 4 processing steps:
  - (1) query decomposition
  - (2) data localization
  - (3) global optimization

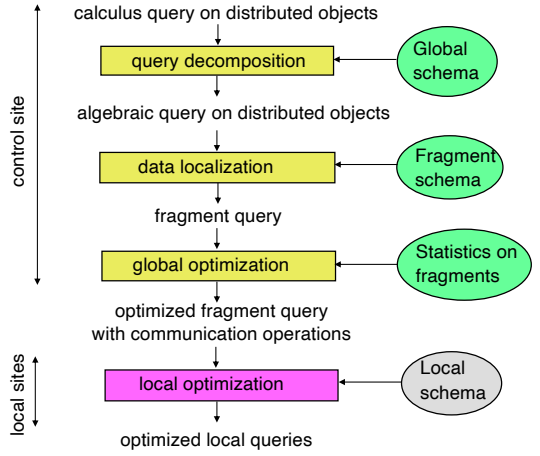
  } control site (uses global information)

  - (4) local optimization — local sites (use local information)

25

25

Generic Layered Scheme for Planning a Dist. Query



This approach was initially designed for distributed relational DBMSs.

It also applies to distributed OODBMSs, Multi-DBMSs, and distributed Multi-DBMSs.

26

26

Distributed Query Optimization

- information needed for optimization (fragment statistics):
  - size of objects, image sizes of attributes
  - transfer costs
  - workload among nodes
  - physical data layout
  - access path, indexes, clustering information
  - properties of the result (objects) - formulas for estimating the cardinalities of operation results
- execution cost is expressed as a weighted combination of I/O, CPU, and communication costs (mostly dominant).

$$\text{total-cost} = C_{CPU} * \#\text{insts} + C_{I/O} * \#\text{I/Os} + C_{MSG} * \#\text{msgs} + C_{TR} * \#\text{bytes}$$

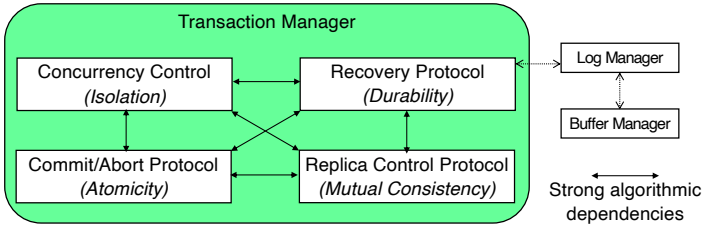
Optimization Goals: response time of single transaction or system throughput

27

27

Distributed Transaction Managment

- Transaction Management (TM) in centralized DBS
  - Achieves transaction ACID properties by using:
    - concurrency control (CC)
    - recovery (logging)
- TM in DDBS
  - Achieves transaction ACID properties by using:

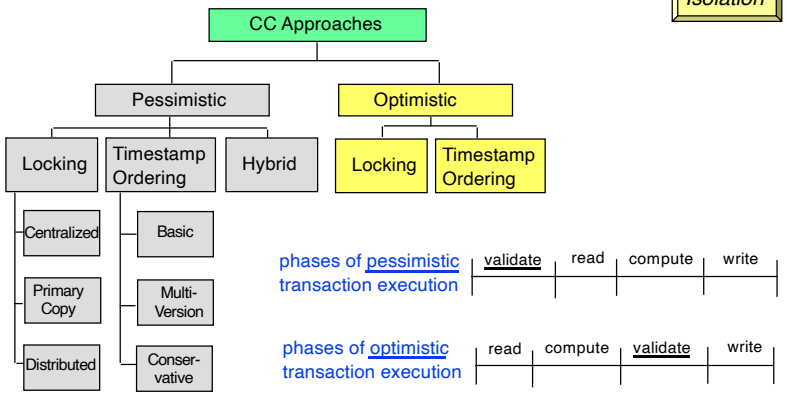


28

28

Classification of Concurrency Control Approaches

Isolation

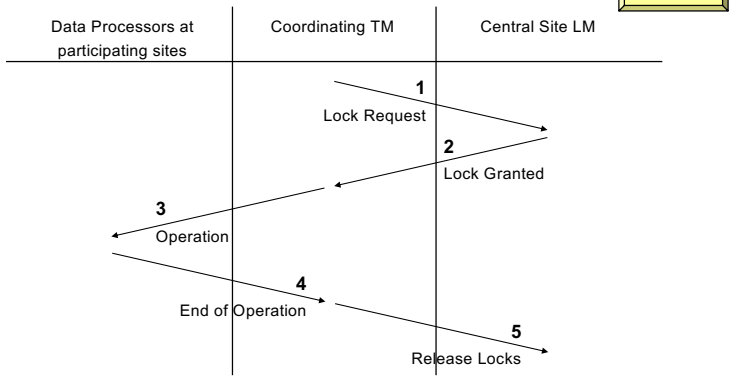


29

29

Communication Structure of Centralized 2PL

Isolation

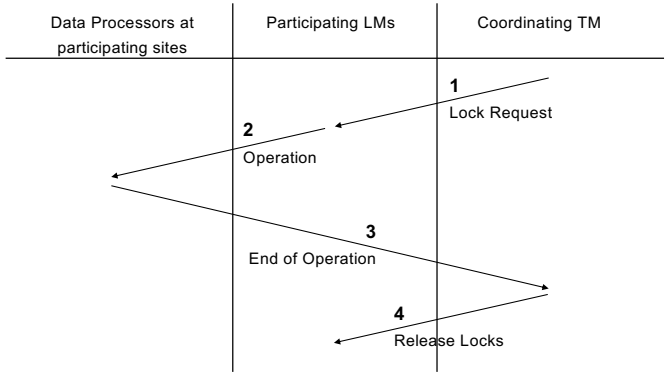


30

30

Communication Structure of Distributed 2PL

Isolation



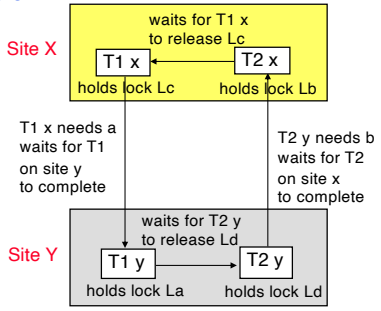
31

31

Distributed Deadlock Management

Isolation

Example



Distributed Waits-For Graph

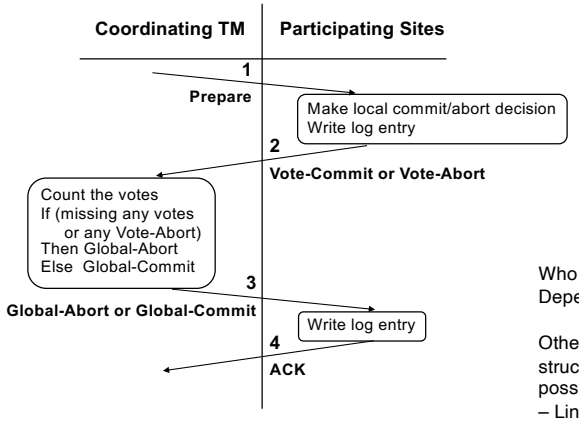
- requires many messages to update lock status
- combine the "wait-for" tracing message with lock status update messages
- still an expensive algorithm

32

32

Communication Structure of Centralized 2P Commit Protocol

Atomicity



Who participates?
Depends on the CC alg.

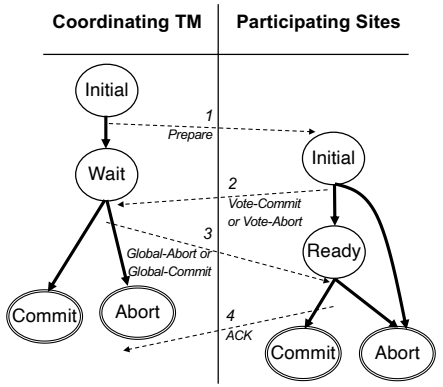
Other communication structures are possible:
– Linear
– Distributed

33

33

State Transitions for 2P Commit Protocol

Atomicity



Advantages:
- Preserves atomicity
- All processes are "*synchronous within one state transition*"

Disadvantages:
- Many, many messages
- If failure occurs, *the 2PC protocol blocks*!

Attempted solutions for the blocking problem:
1) Termination Protocol
2) 3-Phase Commit
3) Quorum 3P Commit

34

34

Failures in a Distributed System

Types of Failure:
- Transaction failure
- Node failure
- Media failure
- Network failure
  - Partitions each containing 1 or more sites

Issues to be addressed:
- How to continue service → *Termination Protocols*
- How to maintain ACID properties while providing continued service → *Modified Concurrency Control & Commit/Abort Protocols*
- How to ensure ACID properties after recovery from the failure(s) → *Recovery Protocols, Termination Protocols, & Replica Control Protocols*

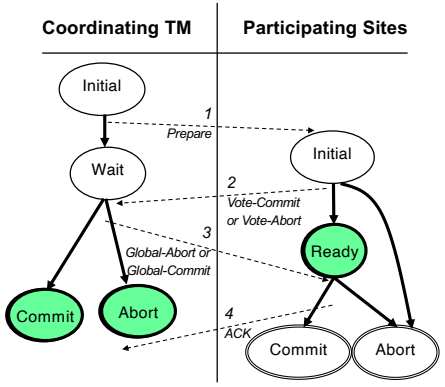
Who addresses the problem?

35

35

Termination Protocols

Atomicity, Consistency



Use timeouts to detect potential failures that could block protocol progress

Timeout states:
Coordinator: wait, commit, abort
Participant: initial, ready

Coordinator Termination Protocol:
Wait – Send global-abort
Commit or Abort – BLOCKED!

Participant Termination Protocol:
Ready – Query the coordinator
If timeout
then query other participants;
If global-abort I global-commit
then proceed and terminate
else BLOCKED!

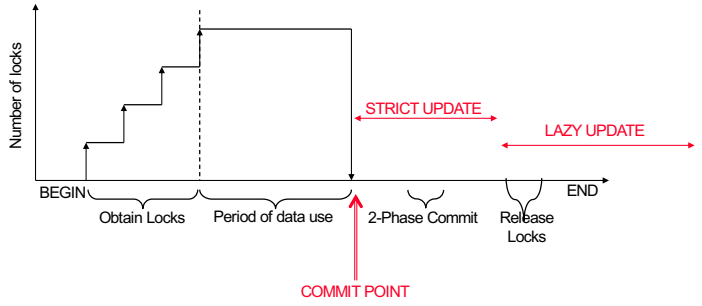
36

36

Replica Control Protocols

Consistency

- Update propagation of committed write operations



37

37

Strict Replica Control Protocol

Consistency

- Read-One-Write-All (ROWA)
- Part of the Concurrency Control Protocol and the 2-Phase Commit Protocol
  - CC locks all copies
  - 2PC propagates the updated values with 2PC messages (or an update propagation phase is inserted between the wait and commit states for those nodes holding an updateable value).

38

38

Lazy Replica Control Protocol

Consistency

- Propagates updates from a primary node.
- Concurrency Control algorithm locks the primary copy node (same node as the primary lock node).
- To preserve single copy semantics, must ensure that a transaction reads a current copy.
  - Changes the CC algorithm for read-locks
  - Adds an extra communication cost for reading data
- Extended transaction models may not require single copy semantics.

39

39

Recovery in Distributed Systems

Atomicity, Durability

Select COMMIT or ABORT (or blocked) for each interrupted subtransaction

Commit Approaches:

Redo – use the undo/redo log to perform all the write operations again

Retry – use the transaction log to redo the entire subtransaction (R + W)

Abort Approaches:

Undo – use the undo/redo log to backout all the writes that were actually performed

Compensation – use the transaction log to select and execute "reverse" subtransactions that semantically undo the write operations.

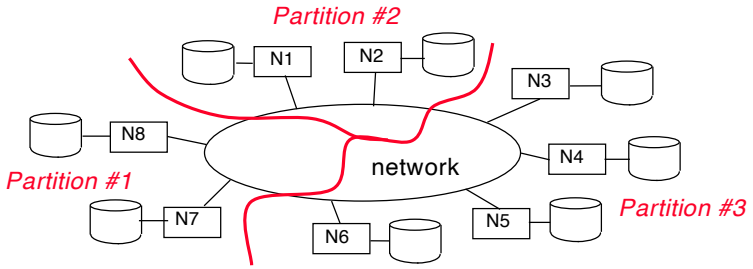
Implementation requires knowledge of:

- Buffer manager algorithms for writing updated data from volatile storage buffers to persistent storage
- Concurrency Control Algorithm
- Commit/Abort Protocols
- Replica Control Protocol

40

40

Network Partitions in Distributed Systems



Issues:

- ✓ Termination of interrupted transactions
- ✓ Partition integration upon recovery from a network failure
- – Data availability while failure is ongoing

41

Data Management for Data Analysis
Data Warehouses

Vera Goebel
Department of Informatics, University of Oslo

Data Analysis (DA) / Machine Learning (ML)
Data Management (DM) for Data Analysis (ML)
OLAP (DA with DBS) versus OLTP (op. DBS)
Data Warehouses (DW)
Data Lakes (DL)

Why Data Analysis?
Can all problems be solved with ML?

Four questions to be answered:

1. Can the problem be clearly defined?
2. Does potentially meaningful data exist?
3. Does the data contain hidden knowledge or useful only for reporting purposes?
4. Will the cost of processing the data be less then the likely increase in profit from the knowledge gained from applying any data analysis?

ML Techniques

- Supervised Learning:
  - Classification
  - Estimation
  - Prediction
- Unsupervised Learning:
  - Clustering
  - Summarization
  - Association

Supervised vs. Unsupervised ML

- Supervised learning (classification, prediction)
  - Supervision: Training data (observations or measurements)
    -> labeled data indicating the class of the data
  - New data is classified based on the training dataset
- Unsupervised learning (summarization, association, clustering)
  - Class labels of training data are unknown
  - Given a set of measurements or observations
    -> establish existence of classes or clusters

ML Types

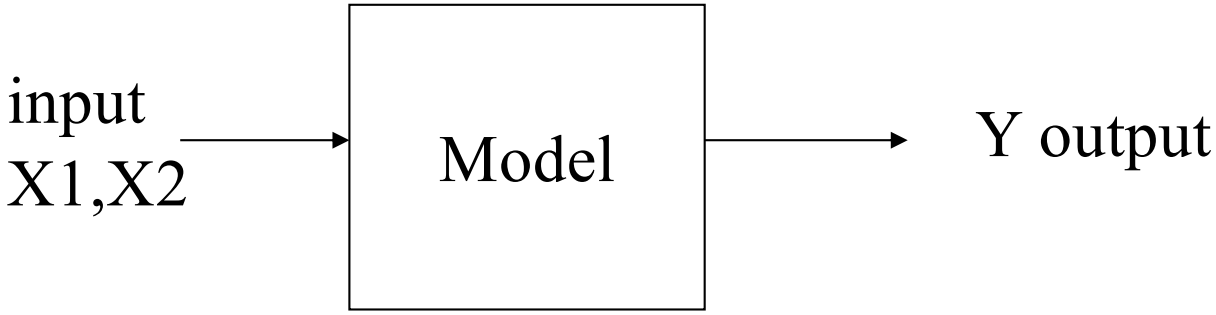
- Descriptive ML
  - characterize the general data properties in DB
  - finds patterns in data
  - user determines which ones are important
- Predictive ML
  - perform inference on the current data to make predictions
  - we know what to predict
- Not mutually exclusive
  - used together
  - descriptive $\rightarrow$ predictive

Descriptive ML

- Discover new patterns in the data
- Used during data exploration
- Typical questions answered:
  - What is in the data?
  - What does it look like?
  - Are there any unusual patterns?
- Patterns at various granularities
- Functionalities of descriptive data mining:
  - Clustering
  - Summarization
  - Visualization
  - Association

Black-box Model

X: vector of independent variables or inputs
Y =f(X) : an unknown function
Y: dependent variables or output
a single variable or a vector



User sees only a black box
Interested in the accuracy of the predictions

Predictive Data Mining

- Using known examples to train model
  - unknown function is *learned from data*
- the more data with known outcomes is available
  - the better the predictive power of the model
- Used to predict outcomes whose inputs are known but the output values are not realized yet
- Never 100% accurate
- Performance of a model on past data is not important
  - to predict the known outcomes
- Performance on unknown data is much more important

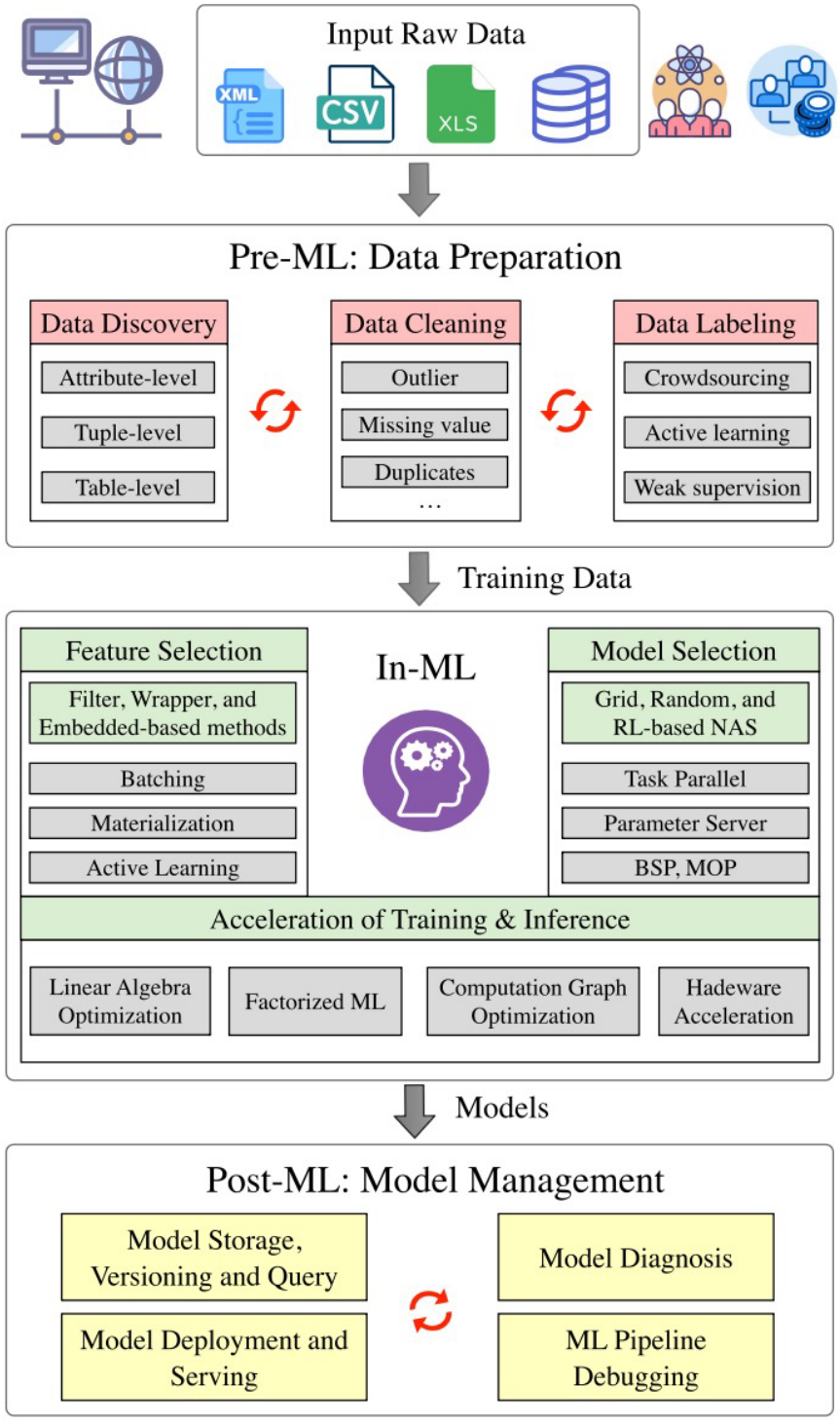
Data Analysis Requirements

- DB schema -> support complex and non-normalized data
  - Summarized and Aggregate data
  - Multiple Relationships
  - Queries must extract multi-dimensional time slices
  - Redundant Data
- Data extraction and filtering -> DB created by extracting data from operational DBS & data imported from external source
  - Need for advanced data extraction & filtering tools
  - Allow batch/scheduled data extraction
  - Support different types of data sources
  - Check for inconsistent data / data validation rules
  - Support advanced data integration / data formatting conflicts
- Analysis tools & user interface
  - advanced data modeling and data presentation tools
- Datasets very large (TB – PB) & advanced hardware

Data Analysis with DW (DB) Why Now?

- Data is being produced -> Big Data!
- Data is being warehoused
- The computing power is available
- The computing power is affordable
- The competitive pressures are strong
- Commercial products are available

Overview - Data Mgnt. for ML



[Chai et al. 2023]

DM Challenges for ML

1. Sufficient high-quality training data is inevitable, data is human expansive to acquire.
2. Large amount of training data and complicated model structures, need end-to-end data analysis pipeline.
3. Given an ML task, need to train many models
   -> hard to manage in real applications

- DB techniques can benefit ML by addressing these challenges.

3 Aspects of ML Pipeline

1. ## Data preparation (Pre-ML):

   Data preparation is the act of manipulating raw data from multiple data sources into a form that can be readily analyzed.

   Tasks: data discovery, data cleaning, data labeling

2. ## Model training and inference (In-ML):

   Improve model performance during training (models become larger and deeper), how to accelerate the entire training process, includes feature selection and model selection.

   Tasks: feature selection, model selection, acceleration of model training & inference

3. ## Model management (Post-ML):

   How to store, query, deploy and debug the models after training. Train iteratively, hard because too much data (parameters, model structures, performance). How to manage these models & artifacts?

   Tasks: model storage, versioning and query; model diagnosis; model deployment and serving; ML pipeline debugging

Pre-ML: Data Preparation

1. Data discovery:
   use data warehouse or data lake to build ML model, we need sufficient data for training and testing
   -> problem: lacking attributes, tuples or features categories: attribute/tuple-level and table-level data discovery.
2. Data cleaning:
   cleaning dirty data, always happens in raw data collected from different resources: missing values, outliers, duplicates, inconsistencies, … -> DW!
3. Data labeling:
   label data for training, need high-quality labels to achieve good ML results -> need human experts, expensive!
   Approaches: crowdsourcing active learning, weak supervision build connections between week labels and downstream ML tasks

In-ML: Model Training & Inference

- Feature extraction:
  batching, materialization (DW!), feature pruning, active learning
- Model selection:
  select most appropriate model or parameters during training
- Computing in training/inference:
  accelerate model training and inference -> DB community: parallelism techniques!

Post-ML: Model Management

- Model storage, versioning and querying:
  store, log, search and analyze model variants and metadata efficiently and effectively: need/use column-store, compression and indexing techniques -> DW!
- Model diagnosis:
  touched model parameters and data artifacts, DM challenges: storage and computation, techniques: sampling, materialization, indexing.
- Model deployment and serving:
  how to support low latency and high throughput model prediction, while guaranteeing accuracy of prediction results.
- ML pipeline debugging:
  (1) data debugging -> learn from data, find root cause of unexpected results, (2) model debugging

Data Analysis Process

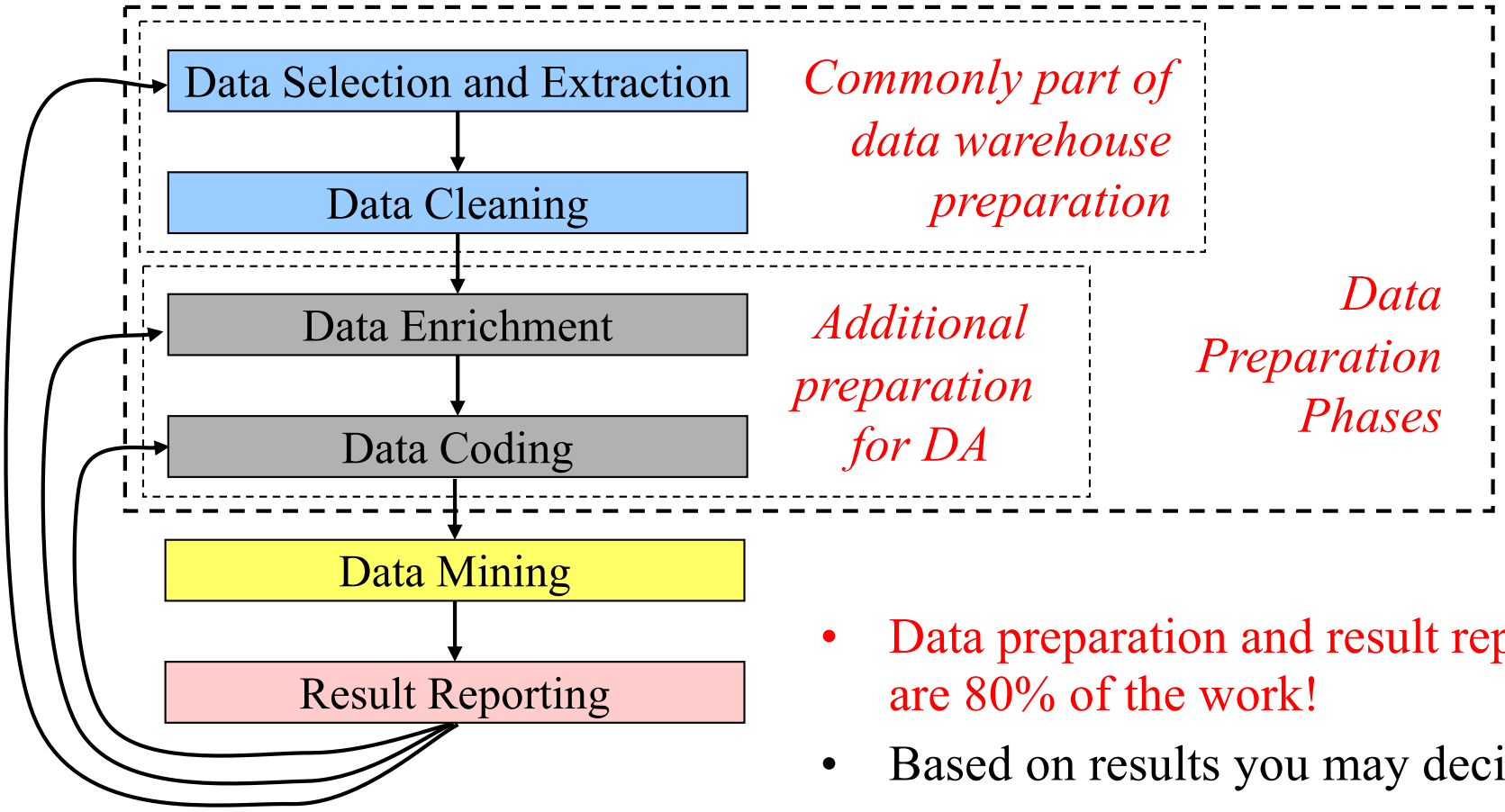
- Problem formulation
- Data collection
  - subset data: sampling might hurt if highly skewed data
  - feature selection: principal component analysis, heuristic search
- Pre-processing: cleaning
  - name/address cleaning, different meanings, duplicate removal, supplying missing values
- Transformation:
  - map complex objects, e.g. time series data, to features, e.g. frequency
- Choosing mining task and mining method
- Result evaluation and visualization

Data Analysis is an iterative process

Data Analysis Process (detailed)

- **1. Goal identification:**
  - Define problem
  - relevant prior knowledge and goals of application
- **2. Creating a target dataset:** data selection
- **3. Data preprocessing:** (80% of effort!) -> good Data Warehouse needed!
  - removal of noise or outliers
  - strategies for handling missing data fields
  - accounting for time sequence information
- **4. Data reduction and transformation:**
  - Find useful features, dimensionality/variable reduction, invariant representation
- **5. Data Mining:**
  - **Choosing functions of data mining:**
    - summarization, classification, regression, association, clustering
  - **Choosing the mining algorithm(s):**
    - which models or parameters
  - **Search for patterns of interest**
- **6. Presentation and evaluation:**
  - visualization, transformation, removing redundant patterns, etc.
- **7. Taking action:**
  - incorporating into the performance system
  - documenting
  - reporting to interested parties

Data Analysis Life Cycle



- Data preparation and result reporting are 80% of the work!
- Based on results you may decide to:
  - Get more data from internal data sources
  - Get additional data from external sources
  - Recode the data

Data Enrichment

- Integrating additional data from external sources
- Sources are public and private agencies
  - Government, Credit bureau, Research lab, …
- Typical data examples:
  - Average income by city, occupation, or age group
  - Percentage of homeowners and car owners by …
  - A person's credit rating, debt level, loan history, …
  - Geographical density maps (for population, occupations, …)
- New data extends each record from internal sources
- Database issues:
  - More heterogenous data formats to be integrated
  - Understand the semantics of the external source data

Data Coding

- Goal: to streamline the data for *effective* and *efficient* processing by the target DA application
- Steps:

  1) Delete records with many missing values
     - But …in fraud detection, missing values are indicators of the behavior you want to discover!

  2) Delete extra attributes
     - Ex: delete customer names if looking at customer classes

  3) Code detailed data values into categories or ranges based on the types of knowledge you want to discover
     - Ex: divide specific ages into age ranges, 0-10, 11-20, …
       map home addresses to regional codes
       convert homeownership to ”yes” or ”no”
       convert purchase date to a month number starting from Jan. 1990

OLTP versus OLAP

- OLTP: On-Line Transaction Processing
  - Processing at operational DBS, short time horizon
  - Optimize throughput (max. # TAs / time unit)
  - Large number of short online TAs
  - ACID transactions
- OLAP: On-Line Analytical Processing
  - Processing at DW, long time horizon
  - Optimize response time (1-n queries / minimal time)
  - Read-only data/queries, periodic refresh/update
  - Complex queries, involve aggregations
  - Store aggregated, historical data in multi-dimensional schemes
  - DW data latency: few hours, Data Marts: 1 day
- 3 aspects: time span, granularity, dimensionality

Aspects

Time span

- Operational DBS / *Data Store:*
  real-time, current transactions, short time frame, specific facts
- Data analysis:
  historic data, long time frame, patterns

Granularity

- Operational DBS / *Data Store:*
  specific Transactions that occur at a given time
- Data analysis:
  shown at different levels of aggregation, different summary levels, decompose (drill down), summarize (roll up)

Dimensionality (Most distinguishing DA characteristic)

- Operational DBS / *Data Store:*
  represents atomic transactions
- Data analysis:
  data is related in many ways, develop the larger picture, multi-dimensional view of data

OLTP versus OLAP

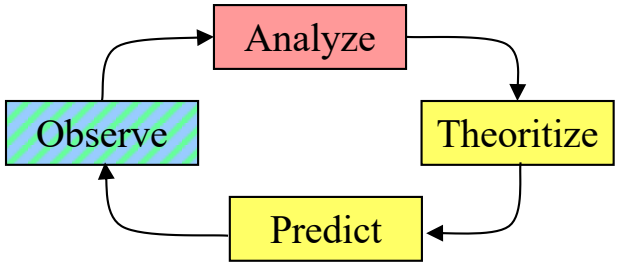
OLTP

- Mostly updates
- Many small transactions
- GB-TB of data
- Raw data
- Clerical users
- Up-to-date data
- Consistency, recoverability critical

OLAP

- Read-only queries
- Periodic updates
- Queries long, complex
- PB-EB+ of data
- Summarized, consolidated data
- ML data scientists

Where SQL Stops Short



ML Tools — *Creates new knowledge in the form of predictions, classifications, associations, time-based patterns*

OLAP Tools — *Analyzes a multi-dimensional view of existing knowledge*

DA-Extended SQL Queries — *Extends the DB schema to a limited, multi-dimensional view*

Complex SQL Queries — *DB schema describes the structure you already know*

Database

- Successful DA uses the entire tool hierarchy

Data Warehouses (DW)

DW is collection of integrated, subject-oriented DBs designed to support data analysis, where each unit of data is non-volatile and relevant to some moment in time [Inmon 1992].

Facts:

- 1990 – IBM, “Business Intelligence”: process of collecting and analyzing
- 1993 – Bill Inmon, “Data Warehouse”
- Growing industry: high RoI (Return of Investment)
- DW solutions offered today by nearly all commercial DBS vendors!
- Expensive to build: 10-1000 million $

What is a Data Warehouse?

- Collection of diverse data
  - subject oriented
  - aimed at executive, decision maker
  - often a copy of operational data
  - with value-added data (e.g., summaries, history)
  - data integrated
  - time variant
  - non-volatile

DW Characteristics

- Subject oriented: data are organized based on how the users refer to them.
- Data integrated: all inconsistencies regarding naming convention and value representations are removed.
- Non-volatile: data are stored in read-only format and do not change over time (except periodic updates/refresh from operational DBS).
- Time variant: data are not current but normally time series.

Data Warehouse (DW)

-> Data Analysis needs multi-dimensional data input

DW: Repository of multiple heterogeneous data sources, organized under a unified multi-dimensional schema at a single site in order to facilitate management decision making.

DW technology includes:

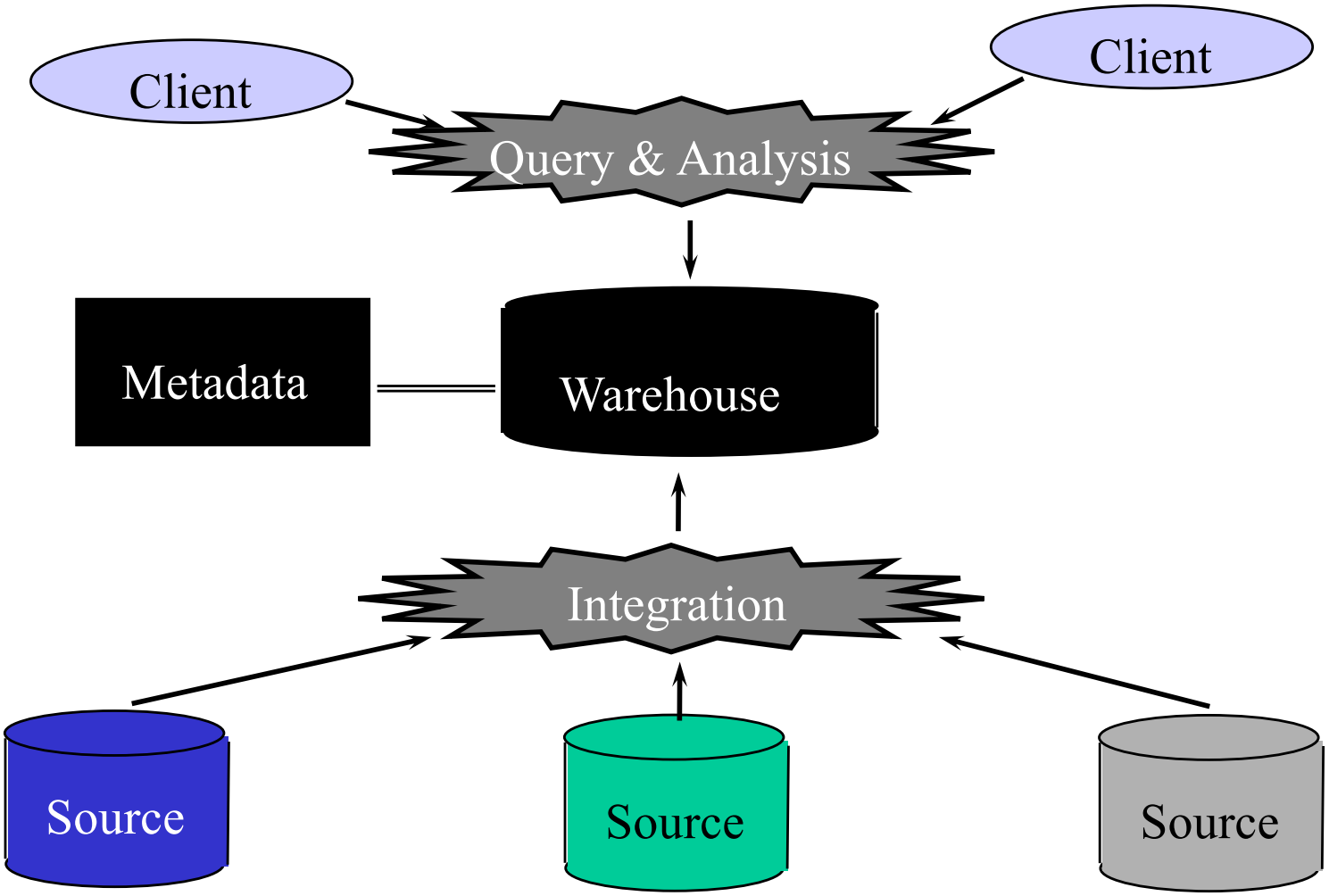
- Data cleaning
- Data integration
- On-Line Analytical Processing (OLAP): Techniques that support multidimensional analysis and decision making with the following functionalities
  - summarization
  - consolidation
  - aggregation
  - view information from different angles
- but additional data analysis tools are needed for
  - classification
  - clustering
  - characterization of data changing over time

DW Characteristics

- Summarized: operational data are mapped into a decision-usable format.
- Large volume: time series data sets are normally quite large.
- Not normalized: DW data can be (often are) redundant.
- Metadata: data about data are stored.
- Data sources: data come from internal and external un-integrated operational systems.

Data Warehouse Architecture

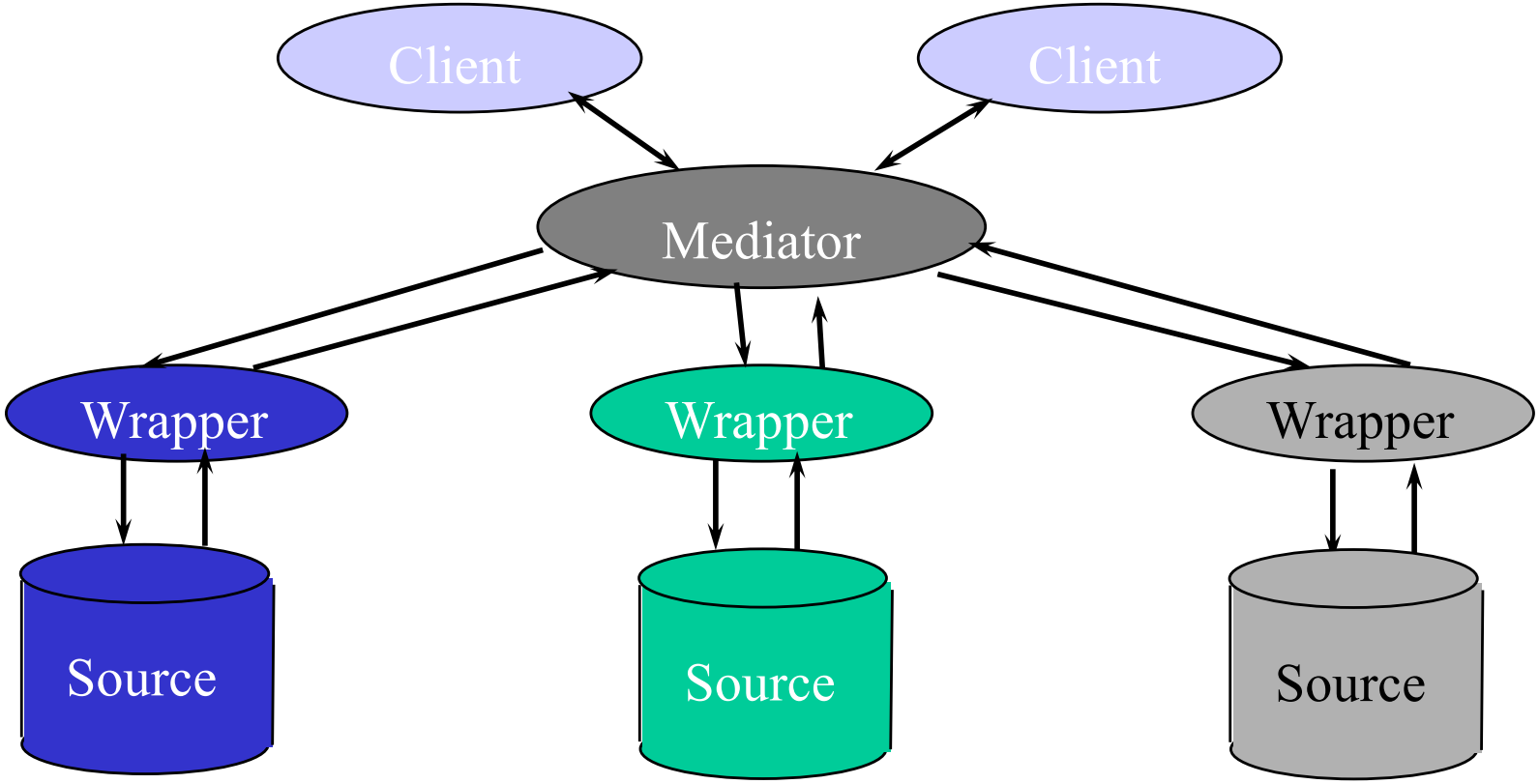
Two Approaches: query-driven (lazy), DW (eager)



Advantages of Data Warehousing

- High query performance
- Queries not visible outside DW
- Local processing at sources unaffected
- Can operate when sources unavailable
- Can query data not stored in a DBMS
- Extra information at warehouse
  - Modify, summarize (store aggregates)
  - Add historical information

Query-Driven Approach



Advantages of Query-Driven Approach

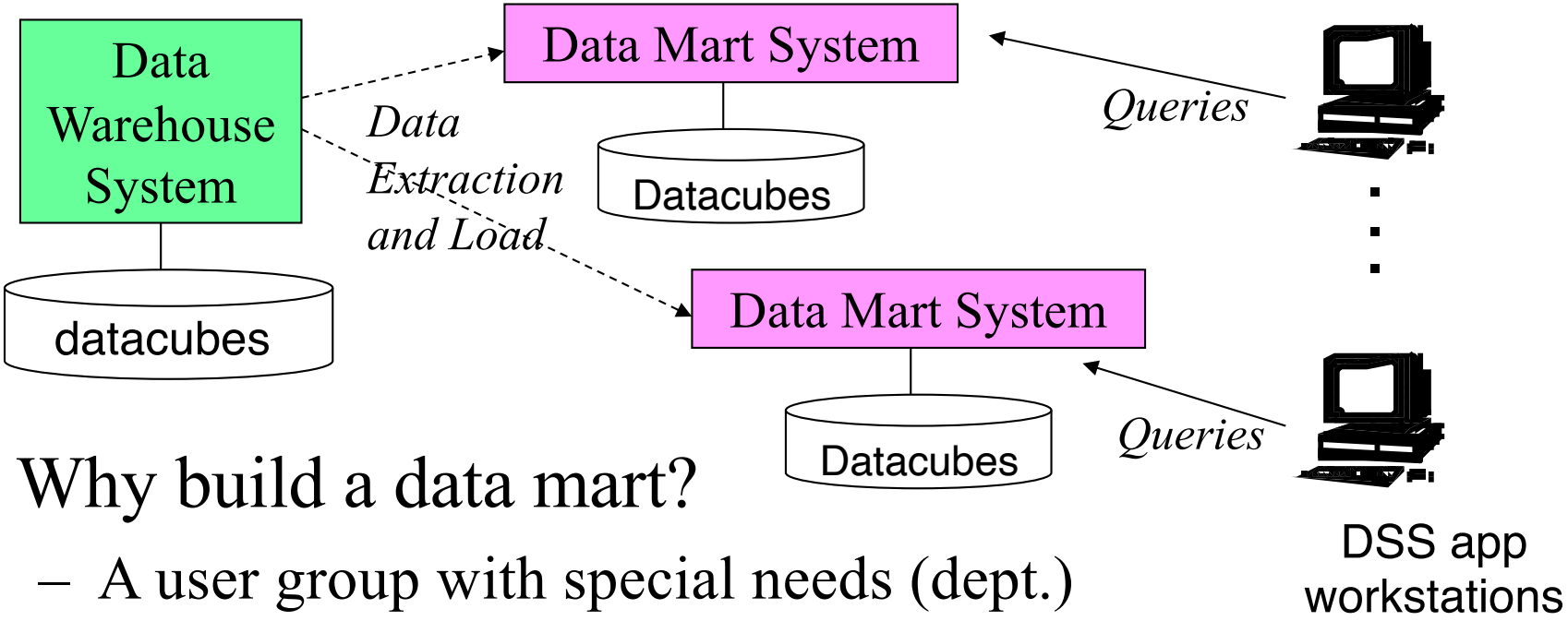
- No need to copy data
  - less storage
  - no need to purchase data
- More up-to-date data
- Query needs can be unknown
- Only query interface needed at sources
- May be less draining on sources

Data Marts

- Smaller data warehouses
- Spans part of organization
  - e.g., marketing (customers, products, sales)
- Do not require enterprise-wide consensus
  - but long term integration problems?

Data Marting

- What: Stores a second copy of a subset of a DW



- Why build a data mart?
  - A user group with special needs (dept.)
  - Better performance accessing fewer records
  - To support a "different" user access tool
  - To enforce access control over different subsets
  - To segment data over different hardware platforms

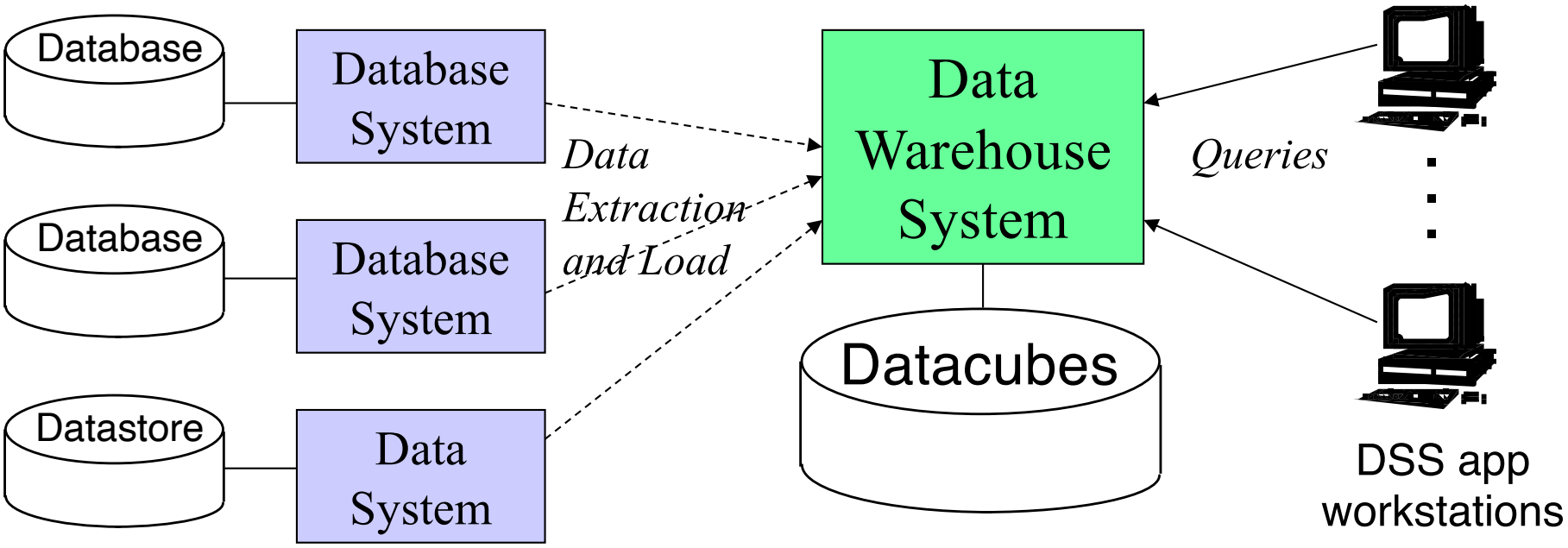
Costs and Benefits of Data Marting

- System costs:
  - More hardware (servers and networks)
  - Define a subset of the global data model
  - More software to:
    - Extract data from the warehouse
    - Load data into the mart
    - Update the mart (after the warehouse is updated)
- User benefits:
  - Define new *measures* not stored in the DW
  - Better performance (mart users and DW users)
  - Support a more appropriate user interface
    - Ex: a browser with forms versus SQL queries
  - Company achieves more reliable access control

DW Models & Operators

- Data Models
  - relations
  - Star schema & snowflake schema
  - Data Cubes
- Operators
  - slice & dice
  - roll-up, drill down
  - pivoting
  - other

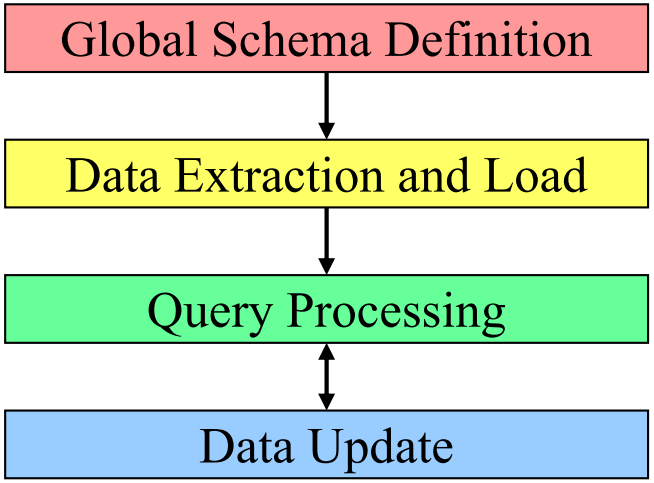
What and Why of Data Warehousing



- What: A very large database containing materialized views of multiple, independent source databases.
  The views generally contain aggregation data (aka datacubes).
- Why: The data warehouse (DW) supports read-only queries for new applications, e.g., ML, OLAP.

DW Life Cycle

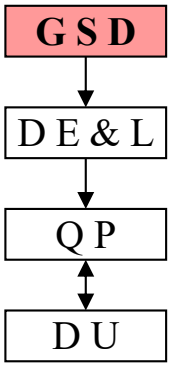
- The Life Cycle:



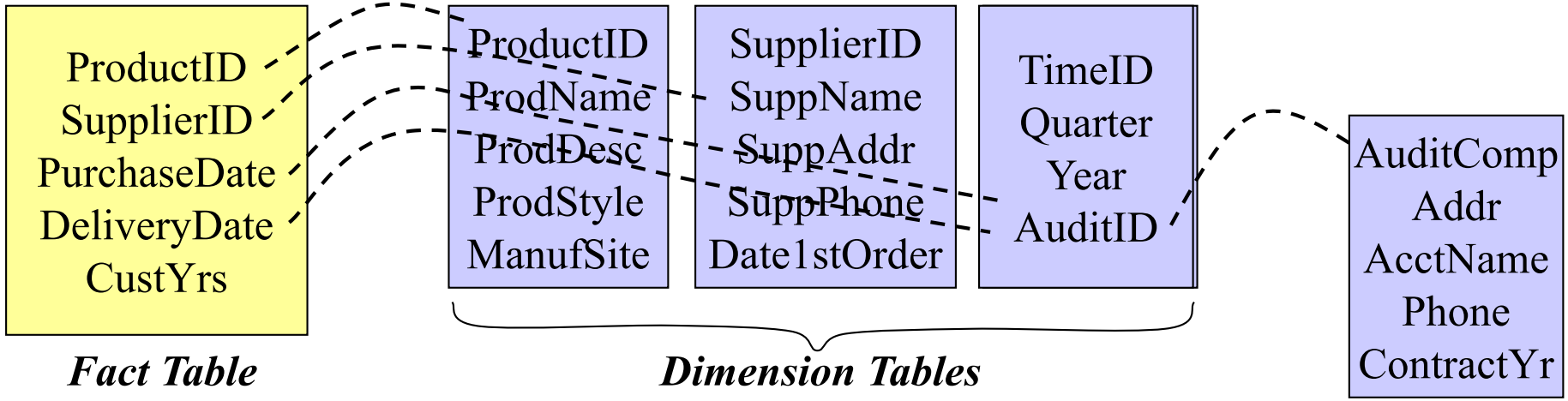
- General Problems:
  - Heavy user demand
  - Problems with source data
    - ownership, format, heterogeneity
  - Underestimating complexity & resources for all phases

- Boeing Computing Services – DW for DSS in airplane repair
  - DW size: 2-3 terabytes
  - Online query services: 24×7 service
  - Data life cycle: retain data for 70+ years (until the airplane is retired)
  - Data update: No "nighttime"; concurrent refresh is required
  - Access paths: Support new and old methods for 70+ years

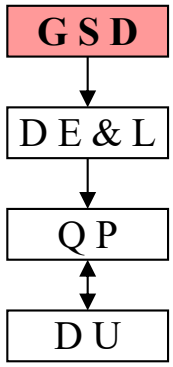
Global Schema Design – Base Tables



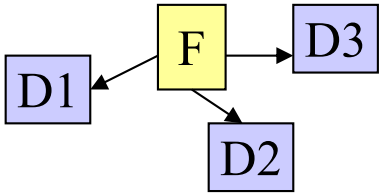
- Fact Table
  - Stores basic facts from the source databases (often denormalized)
  - Data about past events (e.g., sales, deliverings, factory outputs, ...)
  - Have a time (or time period) associated with them
  - Data is very unlikely to change at a data source; no updates
  - Very large tables (up to 1 TB)
- Dimension Table
  - Attributes of one dimension of a fact table (typically denormalized)
  - A chain of dimension tables to describe attributes on other dimension tables, (normalized or denormalized)
  - Data can change at a data source; updates executed occasionally



Schema Design Patterns

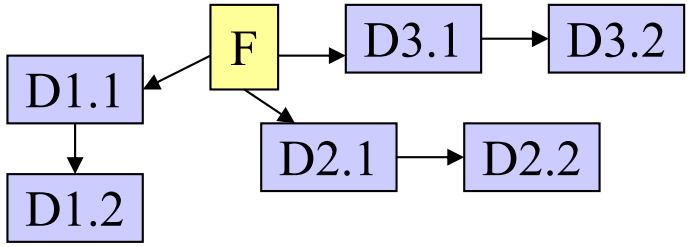


- Star Schema



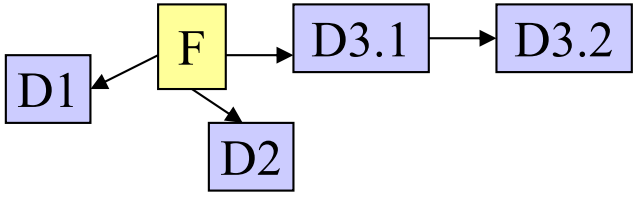
D1, D2, D3 are denormalized

- Snowflake Schema



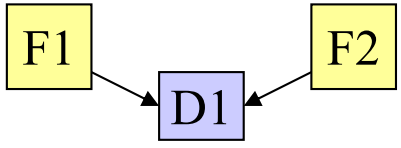
D1, D2, D3 are normalized

- Starflake Schema



D3 may be normalized or denormalized

- Constellation Schema

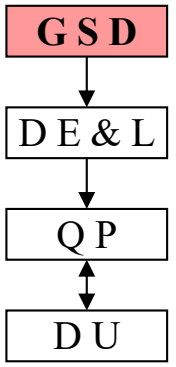


D1 stores attributes about a relationship between F1 and F2

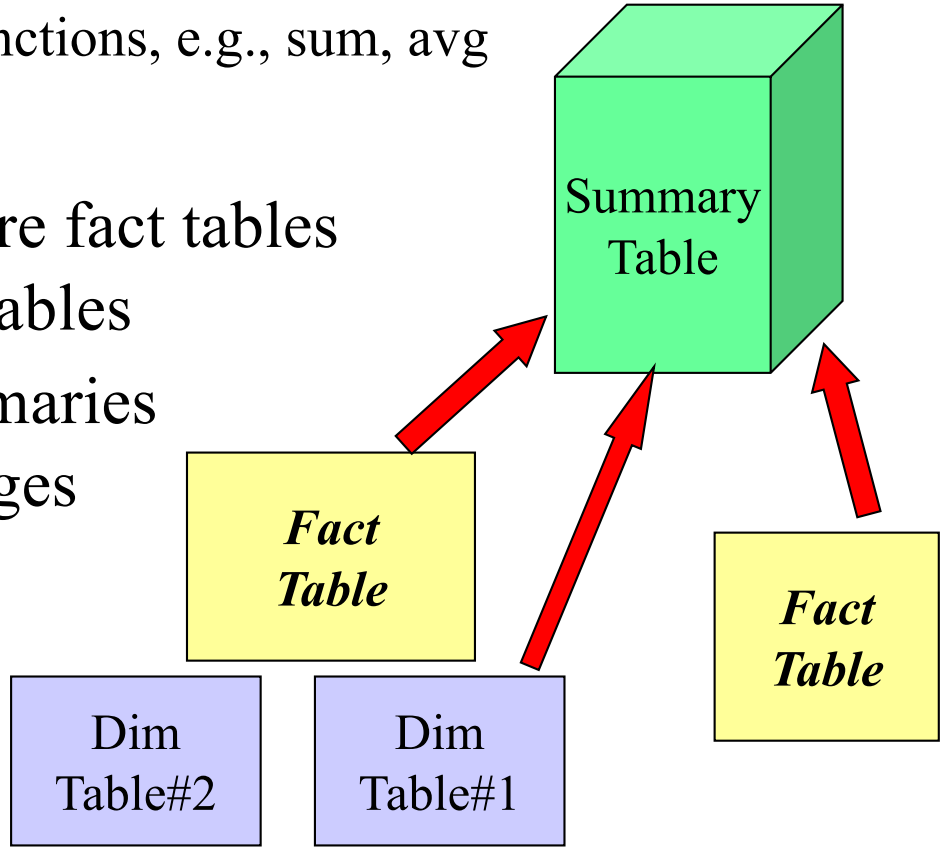
Design Methodologies

- Bottom-up:
  - Data Marts created first
  - Integrate data marts to create DW
  - Bus architecture: collection of conformed dimensions & facts
- Top-down:
  - Use normalized enterprise data model
  - Atomic data, (data at lowest level of detail) stored in DW
  - Dimensional data marts containing data needed for spec. business process, created from DW

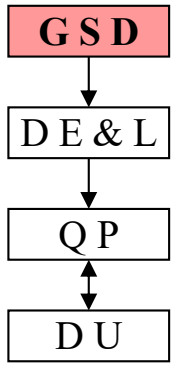
Summary Tables



- aka ***datacubes*** or multidimensional tables
- Store precomputed query results for *likely queries*
  - Reduce on-the-fly join operations
  - Reduce on-the-fly aggregation functions, e.g., sum, avg
- Stores denormalized data
- Aggregate data from one or more fact tables and/or one or more dimension tables
- Discard and compute new summaries as the set of *likely queries* changes



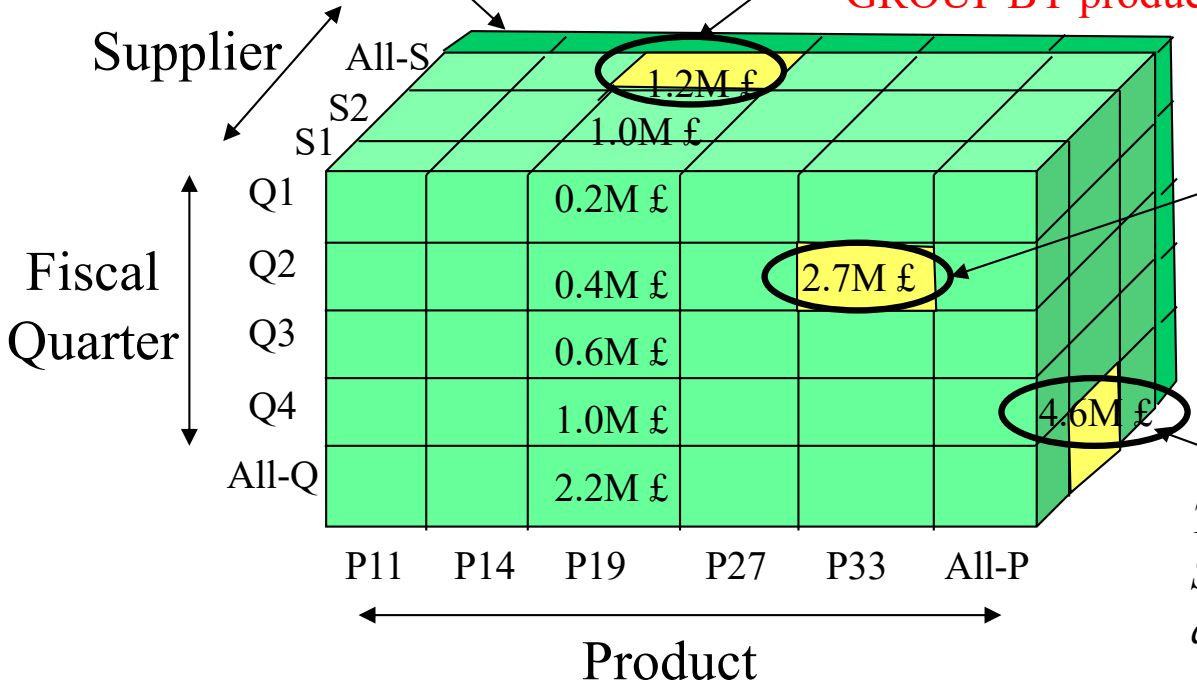
Summary Tables = Datacubes



Total Expenses for Parts by Product, Supplier and Quarter

Average Price paid to all suppliers of parts for Product P11 in the 1st quarter
GROUP BY product, quarter

Total Expenses paid to all suppliers of parts for Product P19 in the 1st quarter
GROUP BY product, quarter

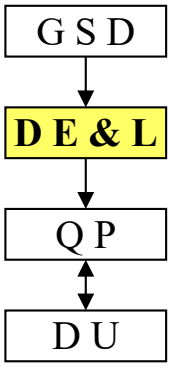


Total Expenses paid to Supplier S1 for parts for Product P33 in 2nd quarter
GROUP BY supplier, product, quarter

Total Expenses paid to Supplier S2 for parts for all products in all quarters
GROUP BY supplier

- Typical, pre-computed *Measures* are:
  - Sum, percentage, average, std deviation, count, min-value, max-value, percentile

Data Extraction and Load



Step1: Extract and clean data from all sources

- Select source, remove data inconsistencies, add default values

Step2: Materialize the views and measures

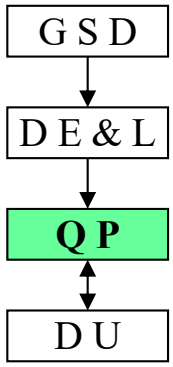
- Reformat data, recalculate data, merge data from multiple sources, add time elements to the data, compute measures

Step3: Store data in the DW

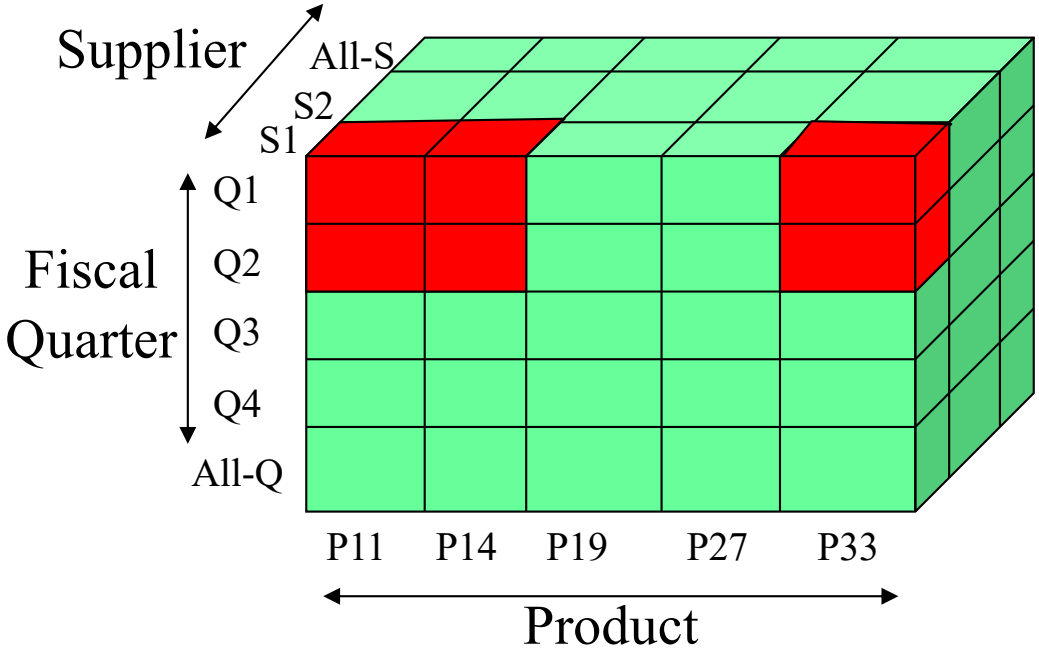
- Create metadata and access path data, such as indexes

- Major Issue: ***Failure during extraction and load***
- Approaches:
  - UNDO/REDO logging
    - Too expensive in time and space
  - Incremental Checkpointing
    - When to checkpoint? Modularize and divide the long-running tasks
    - Must use UNDO/REDO logs also; Need high/performance logging

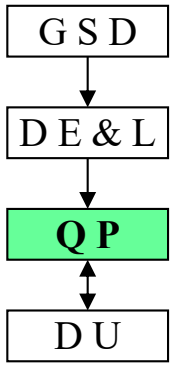
User Queries



- Retrieve pre-computed data or formulate new measures not materialized in the DW.
- New user operations on logical datacubes:
  - Roll-up, Drill-down, Pivot/Rotate
  - Slicing and Dicing with a "data blade"
  - Sorting
  - Selection
  - Derived Attributes

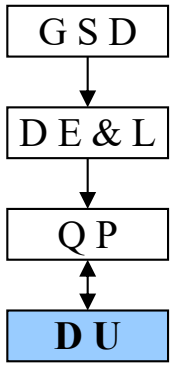


Query Processing



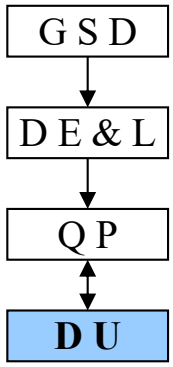
- Traditional query transformations
- Index intersection and union
- Advanced join algorithms
- Piggy-backed scans
  - Multiple queries with different selection criteria
- SQL extensions => new operators
  - Red Brick Systems has proposed 8 extensions, including:
    - MovingSum and MovingAvg
    - Rank … When
    - RatioToReport
    - Tertiles
    - Create Macro

Data Update



- Data sources change over time
- Must "refresh" the DW
  - Adds new historical data to the fact tables
  - Updates descriptive attributes in the dimension tables
  - Forces recalculation of measures in summary tables
- Issues:
  1. Monitoring/tracking changes at the data sources
  2. Recalculation of aggregated measures
  3. Refresh typically forces a shutdown for DW query processing

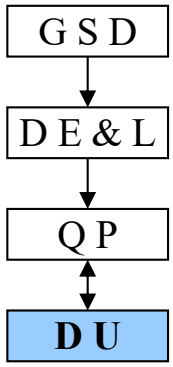
Monitoring Data Sources



Approaches:

1. Value-deltas - Capture before and after values of all tuples changed by normal DB operations and store them in differential relations.
   - Issues: must take the DW offline to install the modified values
2. Operation-deltas – Capture SQL updates from the transaction log of each data source and build a new log of all transactions that effect data in the DW.
   - Advantages: DW can remain online for query processing while executing data updates (using traditional concurrency control)
3. Hybrid – use value-deltas and operation-deltas for different data sources or a subset of the relations from a data source.

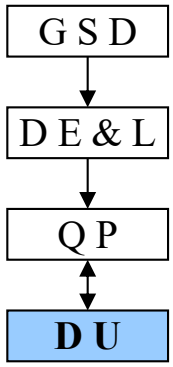
Creating a Differential Relation



Approaches at the Data Source:

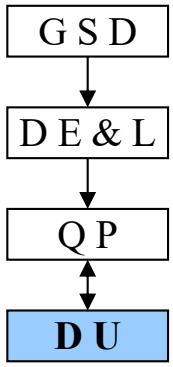
1. Execute the update query 3 times
   - (1) Select and record the before values;
     (2) Execute the update;
     (3) Select and record the after values
   - Issues: High cost in time & space;
     reduces autonomy of the data sources
2. Define and insert DB triggers
   - Triggers fire on "insert", "delete", and "update" operations; Log the before and after values
   - Issues: Not all data sources support triggers;
     reduces autonomy of the data sources

Creating Operation-Deltas



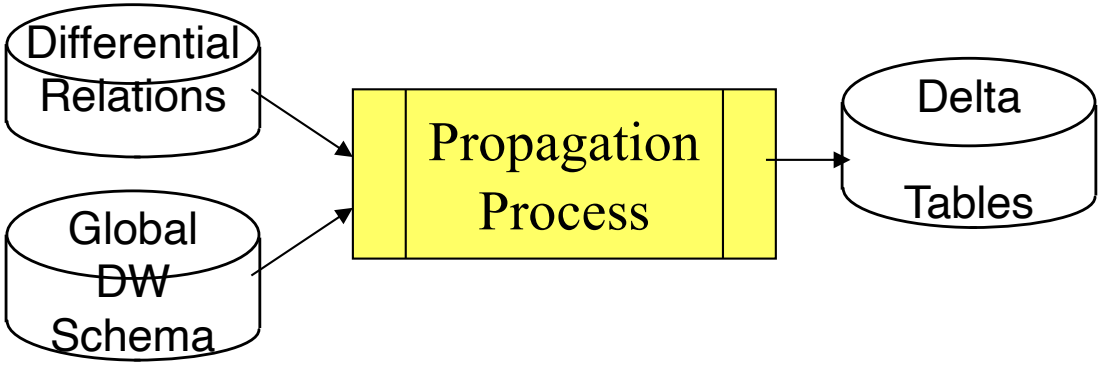
- The process:
  - Scan the transaction log at each data source
  - Select pertinent transactions and delta-log them
- Advantage:
  - Op-delta is much smaller than the value-delta
- Issues:
  - Must transform the update operation on the data source schema into an update operation on the DW schema – not always possible.
    Hence can not be used in all cases.

Recalculating Aggregated Measures



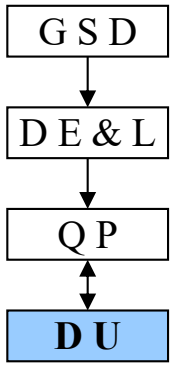
- Delta Tables
  - Assume we have differential relations for the base facts in the data sources (i.e., value deltas)
  - Two processing phases (Propagation & Refresh):

    1) Propagation – pre-compute all new tuples and all replacement tuples and store them in a delta table

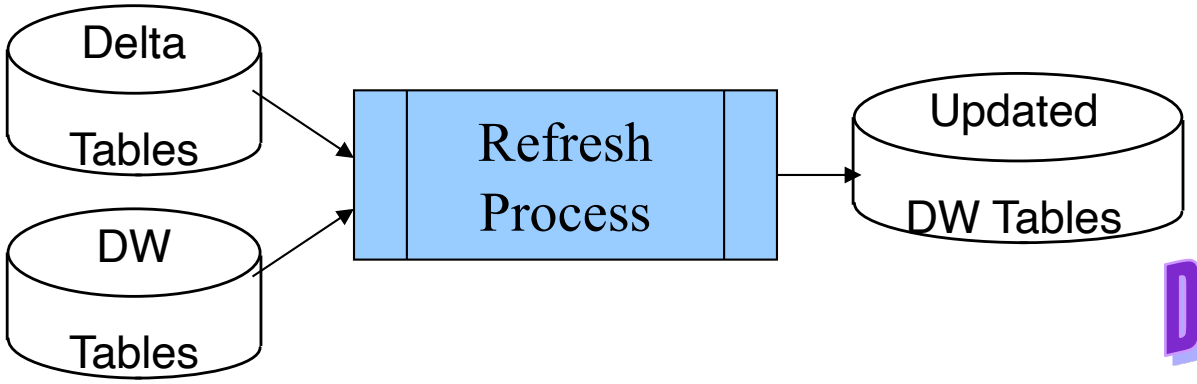


DW is online in phase #1

Recalculate Aggregated Measures



2) Refresh – Scan the DW tuples, replace existing tuples with the pre-computed tuple values, insert new tuples from the delta tables



DW is offline
ONLY in phase #2

Issue:
Can not pre-compute Delta Table for non-commutative measures
Ex: average (without #records), percentiles
Must compute these during the refresh phase.

Data Lake (DL)

- Collection of multi-modal data stored in their raw formats.
  - Each element has a unique identifier and metadata
  - For each business question, you can find the relevant data set to analyze it
- Often based on Hadoop software
  - Enterprise Hadoop

DL Advantages

- Schema on read
  - Write the data as they are, read them according to a diagram (e.g. code of the Map function)
  - More flexibility, multiple views of the same data
- Multi-workload data processing
  - Different types of processing on the same data
  - Interactive, batch, real time
- Cost-effective data architecture
  - Excellent cost/performance and RoI ratio with SN cluster and open source technologies

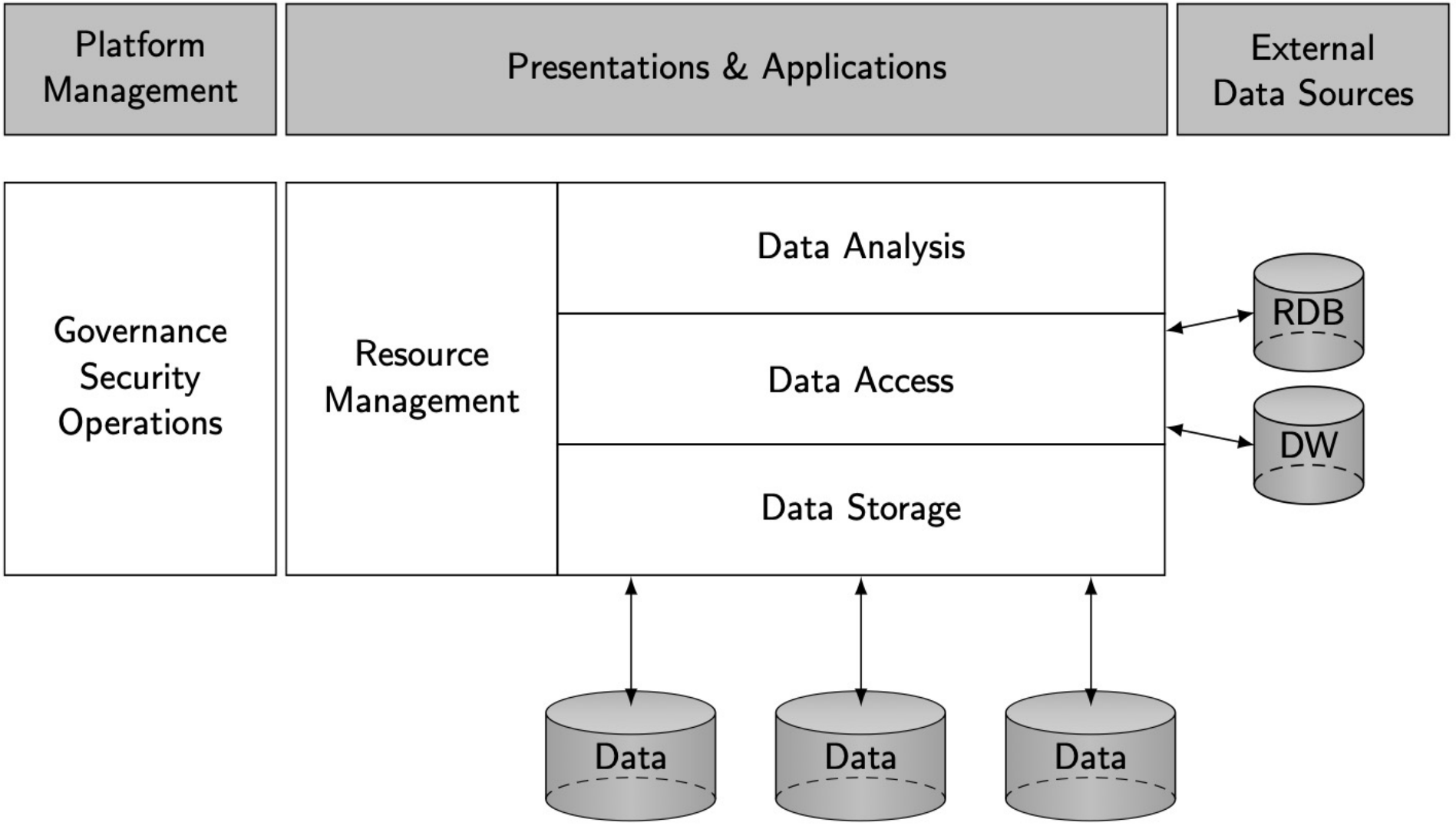
DL Principles

- Collect all useful data
  - Raw data, transformed data
- Dive from anywhere
  - Users from different business units can explore and enrich the data
- Flexible access
  - Different access paths to shared infrastructure
    - Batch, interactive (OLAP and BI), real-time, search,.....

Main Functions

- Data management, to store and process large amounts of data
- Data access: interactive, batch, real time, streaming
- Governance: load data easily, and manage it according to a policy implemented by the *data steward*
- Security: authentication, access control, data protection
- Platform management: provision, monitoring and scheduling of tasks (in a cluster)

DL Architecture



DL versus DW

Data Lake

- Data from het. various data sources
- Multi-workload processing
- Shorter development process
- Storing data objects as: id, metadata, original data
- No cleaning or transformation
- Schema-on-read
- Cost-effective architecture
- Easy to adapt & extend to changes
- Problems: acquire metadata, no data cleaning, consistency & integration, QP&opt., data quality, governance, security, …

Data Warehouse

- Physical DB integration
- ML (OLAP) workloads
- Long development process
- Upfront precise global schema and data modeling
- DB consistency
- Schema-on-write
- Query processing (QP) efficient
- Complex ETL process: extract, transform (clean, …), load
- Difficult & costly to adapt to changes
- Difficult schema and data updates

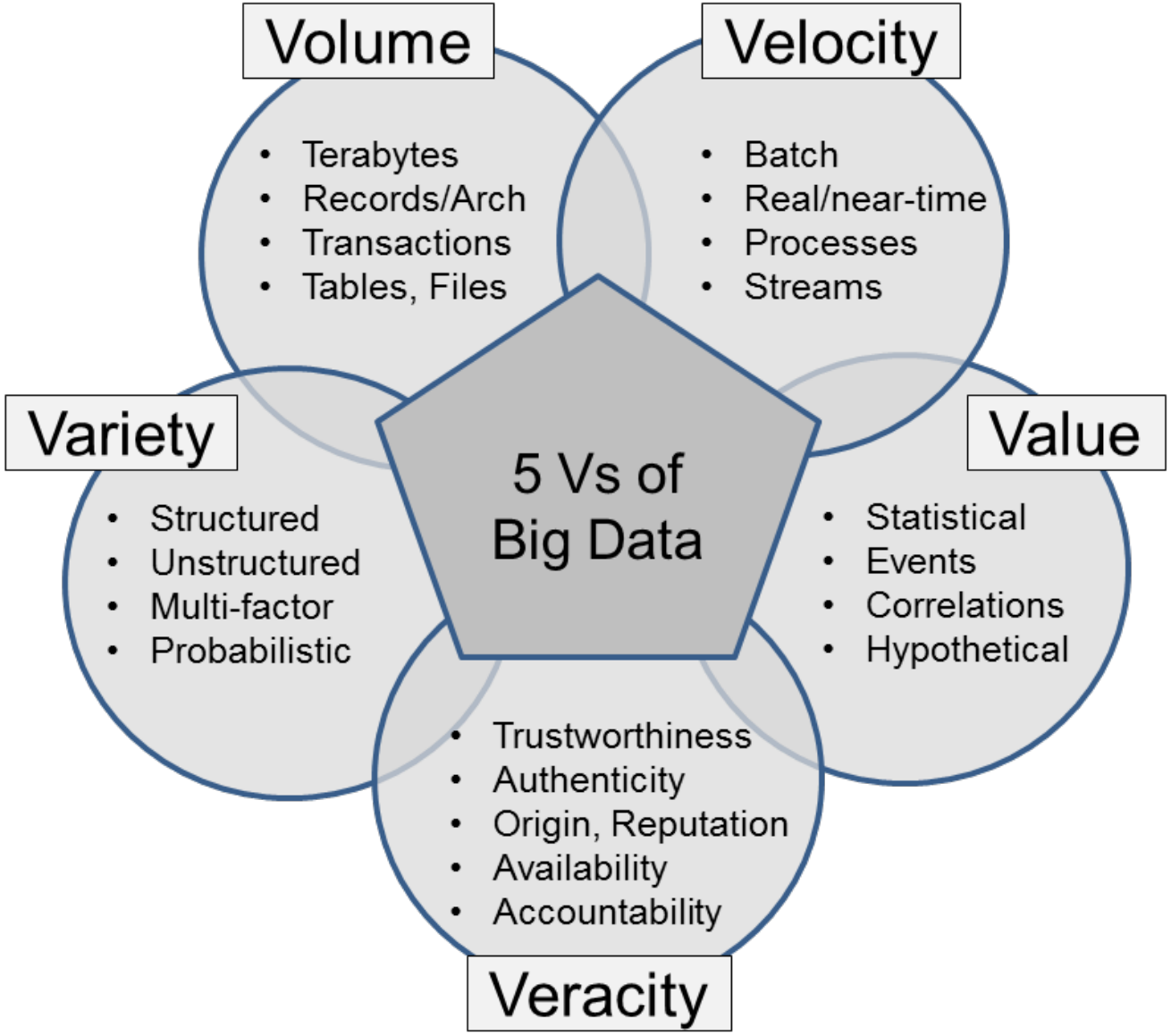
Scalable Data Management
Cloud Data Management

Vera Goebel

Thomas Plagemann

Big Data Management & Processing
Cloud Computing
CAP Theorem
NoSQL, NewSQL, PolyStores
Scalable DSP
Lazy Migration

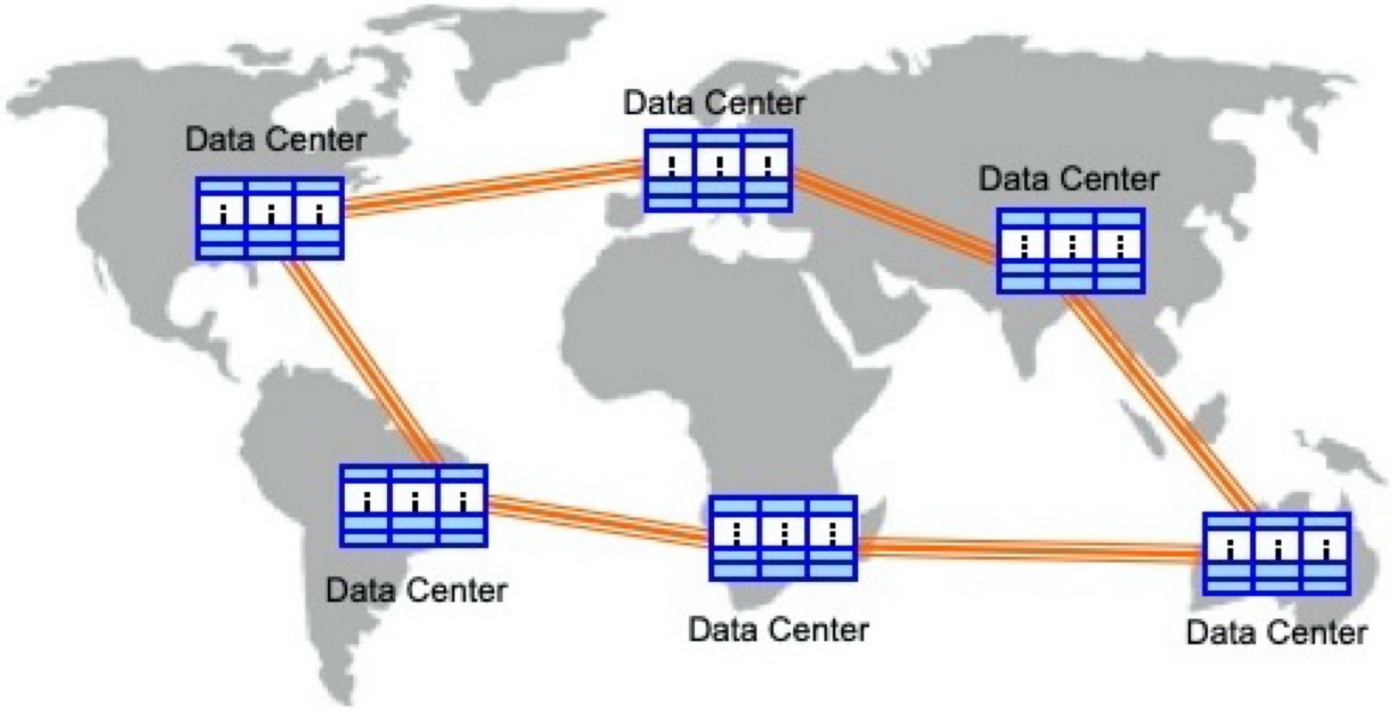
Big Data – Main Characteristics



Five Vs

- Volume
  - Increasing data size: petabytes (10^{15}) to zettabytes (10^{21})
- Variety (heterogeneity)
  - 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, consistency, data trustworthiness
- Value
  - Data analysis
  - Added-value to data

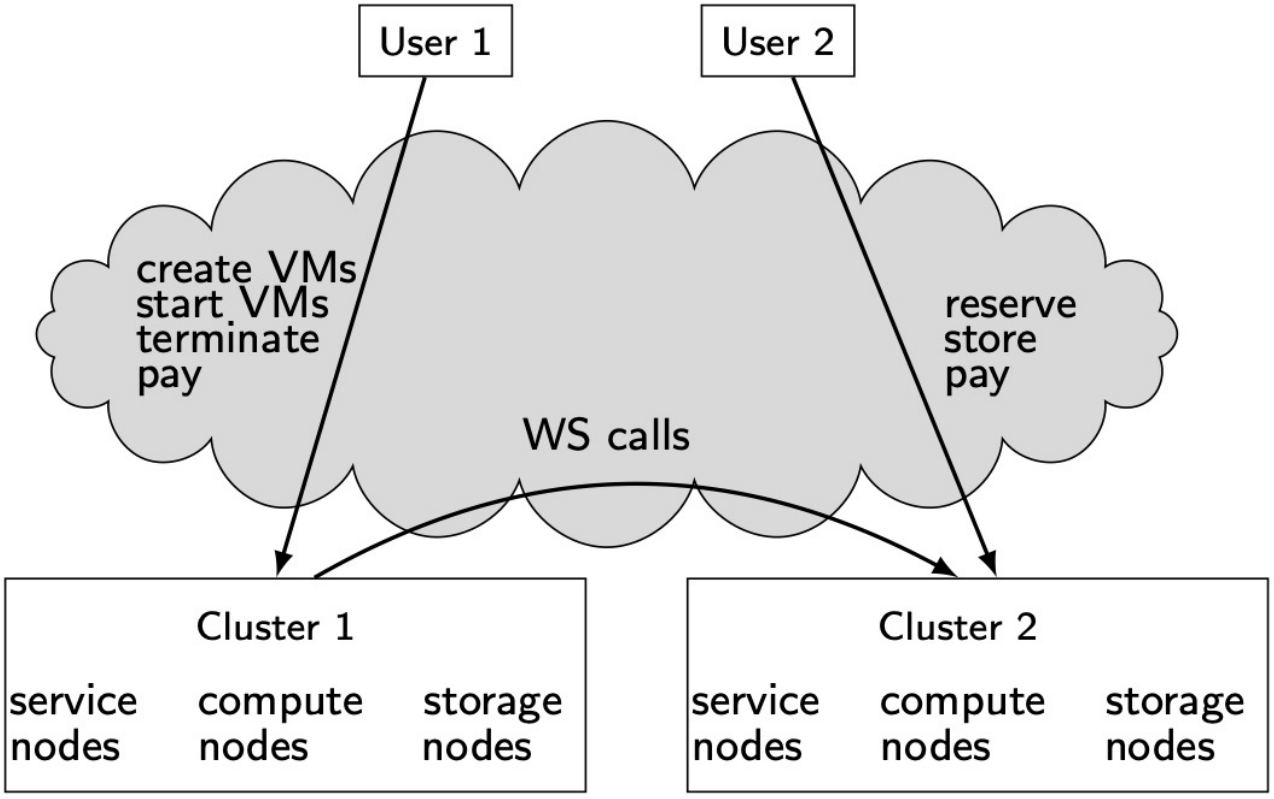
Geo-Distributed Data Centers



Cloud Computing & Architecture

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



Cloud Computing

On-demand, reliable services provided over the Internet in a cost-efficient manner (use Data Centers)

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

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

Cloud Taxonomy

- Infrastructure-as-a-Service (IaaS)
  - Computing, networking and storage resources, as a service
  - Provides elasticity: ability to scale up (add more resources) or scale down (release resources) as needed
  - Example: Amazon Web Services: Virtual machine EC2
- Software-as-a-Service (SaaS)
  - Application software as a service
  - Generalizes the earlier ASP model with tools to integrate other applications, e.g. developed by the customer (using the cloud platform)
  - Hosted applications: from simple (email, calendar) to complex (CRM, data analysis or social network)
  - Example: Safesforce CRM system
- Platform-as-a-Service (PaaS)
  - Computing platform with development tools and APIs as a service
  - Enables developers to create and deploy custom applications directly on the cloud infrastructure and integrate them with applications provided as SaaS
  - Example: Google Apps

Cloud Characteristics

- Compute power is elastic, but only if workload is parallelizable! -> Applications must be designed to run on shared-nothing architecture
- Data is stored at untrusted hosts! Security and Privacy?
- Data is replicated, often across large geographic distances -> Transactions? Consistency? Security and Privacy?

Main Issue: Security and Privacy

- Current solutions
  - Internal cloud (or private cloud): the use of cloud technologies but in a private network behind a firewall
    - Much tighter security
    - Reduced cost advantage because the infrastructure is not shared with other customers (as in public cloud)
    - Compromise: hybrid cloud (internal cloud for OLTP + public cloud for OLAP)
  - Virtual private cloud: Virtual Private Network (VPN) within a public cloud with security services
    - Promise of a similar level of security as an internal cloud and tighter integration with internal cloud security
    - *But such security integration is complex and requires talented security administrators*

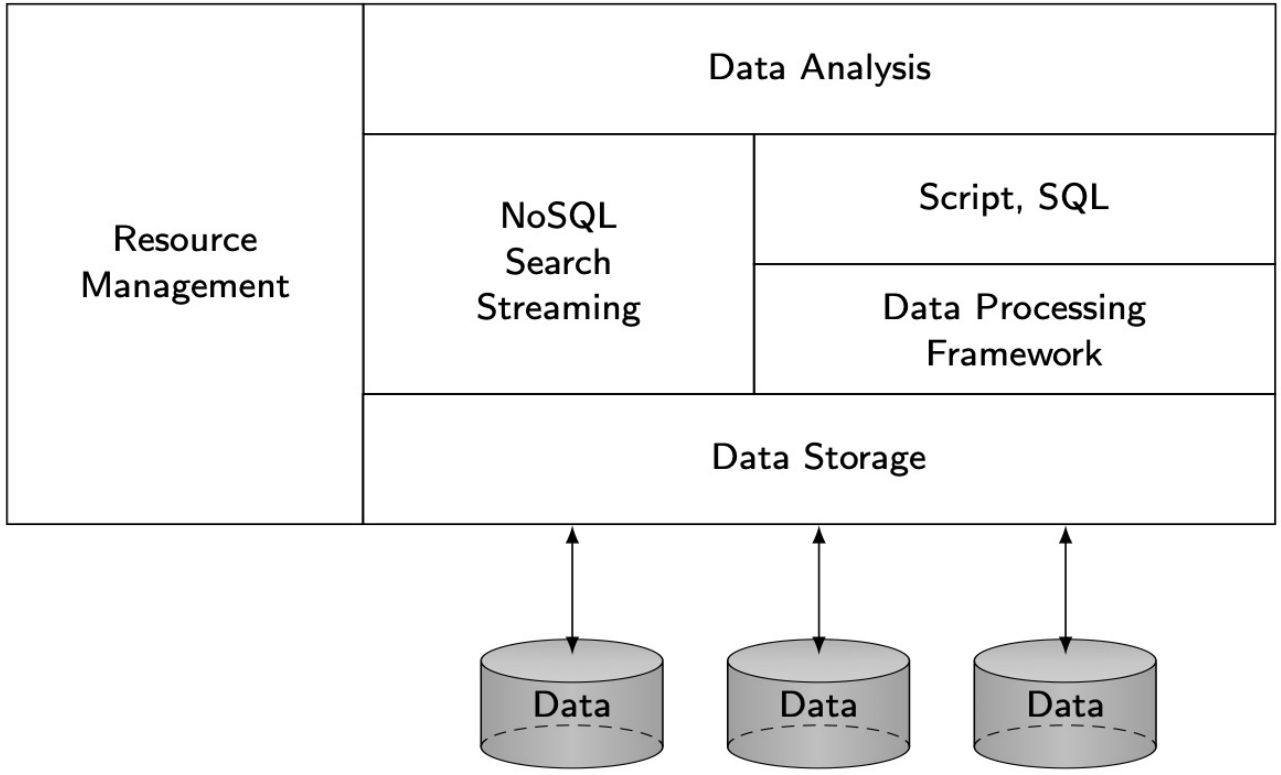
Data Management in the Cloud

- Data management applications are potential candidates for deployment in the cloud
  - **industry:** enterprise database system have significant up-front cost that includes both hardware and software costs
  - **academia:** manage, process and share mass-produced data in the cloud
- Many “Cloud Killer Apps” are in fact data-intensive
  - Batch Processing as with map/reduce
  - Online Transaction Processing (OLTP) as in automated business applications
  - Online Analytical Processing (OLAP) as in data mining or machine learning (Data Warehouses)

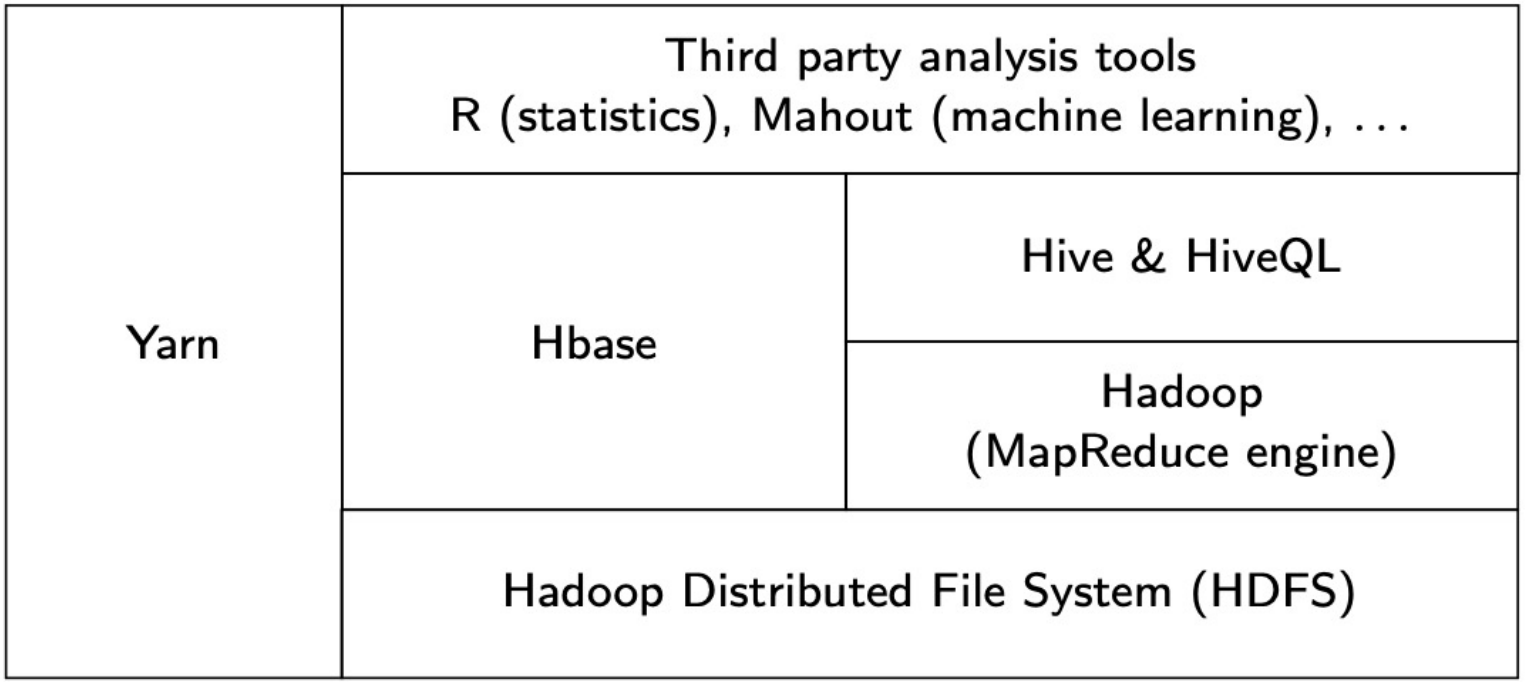
Big Data Processing Platforms

- Applications that do not need full DBMS functionality
  - Data analysis of very large data sets
  - Highly dynamic, irregular, schemaless, …
- Parallelization problems
- MapReduce/Spark
- Advantages
  - Flexibility
  - Scalability
  - Efficiency
  - Fault-tolerance
- Disadvantage
  - Reduced functionality
  - Increased programming effort

Big Data Software Stack



Hadoop Stack



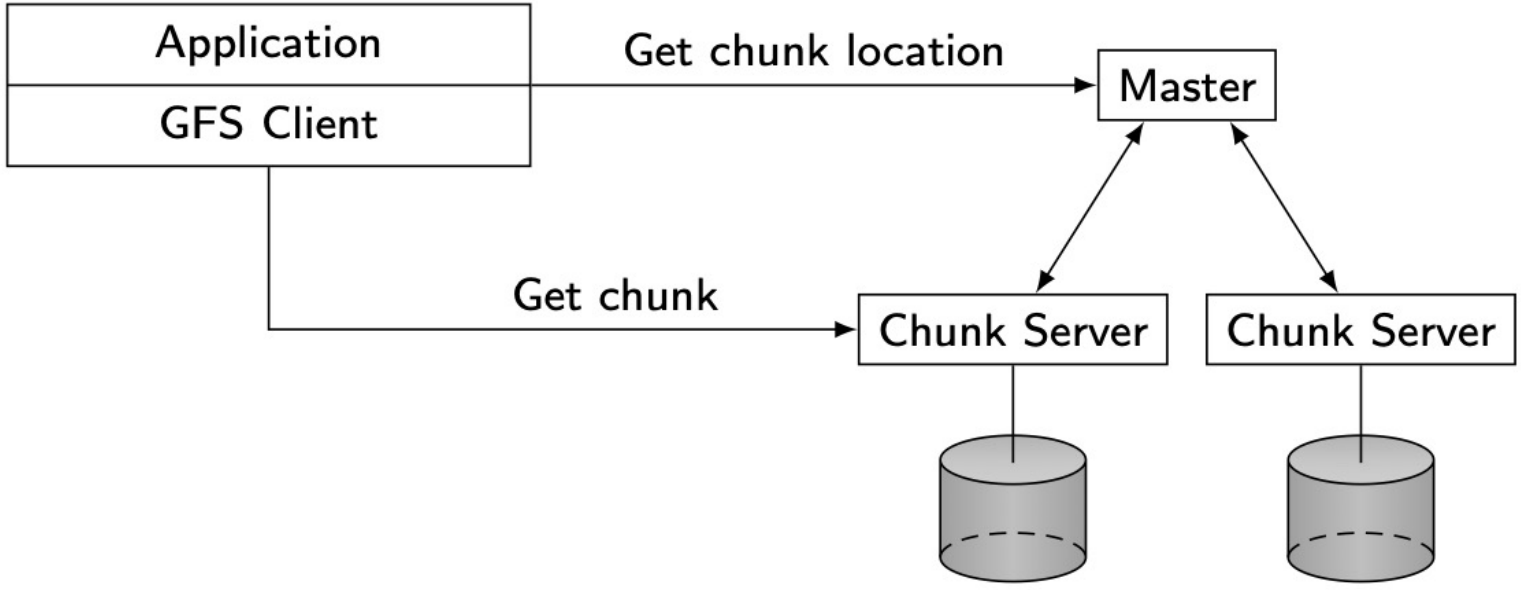
Distributed Storage System

Storing and managing data across the nodes of shared-nothing cluster

- Object-based
  - Object = ⟨oid, data, metadata⟩
  - Metadata can be different for different object
  - Easy to move
  - Flat object space → billions/trillions of objects
  - Easily accessed through REST-based API (`get`/`put`)
  - Good for high number of small objects (photos, mail attachments)
- File-based
  - Data in files of fixed- or variable-length records
  - Metadata-per-file stored separately from file
  - For large data, a file needs to be partitioned and distributed

Google File System (GFS)

- Targets shared-nothing clusters of thousands of machines
- Targets applications with characteristics:
  - Very large files (several gigabytes)
  - Mostly read and append workloads
  - High throughput more important than low latency
- Interface: `create`, `open`, `read`, `write`, `close`, `delete`, `snapshot`, `record append`



Scalable Data Management

- Horizontal Scaling
  -> performance and scalability – throughput (OLTP) / data analysis (DW)
  -> parallel data access/storage/processing
- Shared Nothing architectures
- Applications: big web players, IoT, Data Analysis (DW)
- Big data: (often) unstructured data
- Data interconnexion: Hyperlinks, tags, blogs, …
- Very high scalability: Data size, numbers of users
- Limits of relational DBMSs (SQL)
  - Need for skilled DBA and well-defined schemas
  - SQL and complex tuning
  - Hard to make updates scalable
    - Parallel RDBMS use a shared-disk for OLTP
- CAP Theorem - a shared-data system can only choose at most 2 out of 3 properties: Consistency, Availability, and Tolerance to Partitions

CAP Theorem

- Polemical topic
  - “A database cannot provide consistency AND availability during a network partition”.
  - Argument used by NoSQL to justify their lack of ACID properties
  - But has nothing to do with scalability
- Two different points of view
  - Relational databases
    - Consistency is essential
      - ACID transactions
  - Distributed systems
    - Service availability is essential
      - Inconsistency tolerated by the user, e.g. web cache

What is the CAP Theorem?

- Desirable properties of a distributed system
  - ***C****onsistency*: all nodes see the same data values at the same time
  - ***A****vailability*: all requests get an answer
  - ***P****artition tolerance*: the system keeps functioning in case of network failure

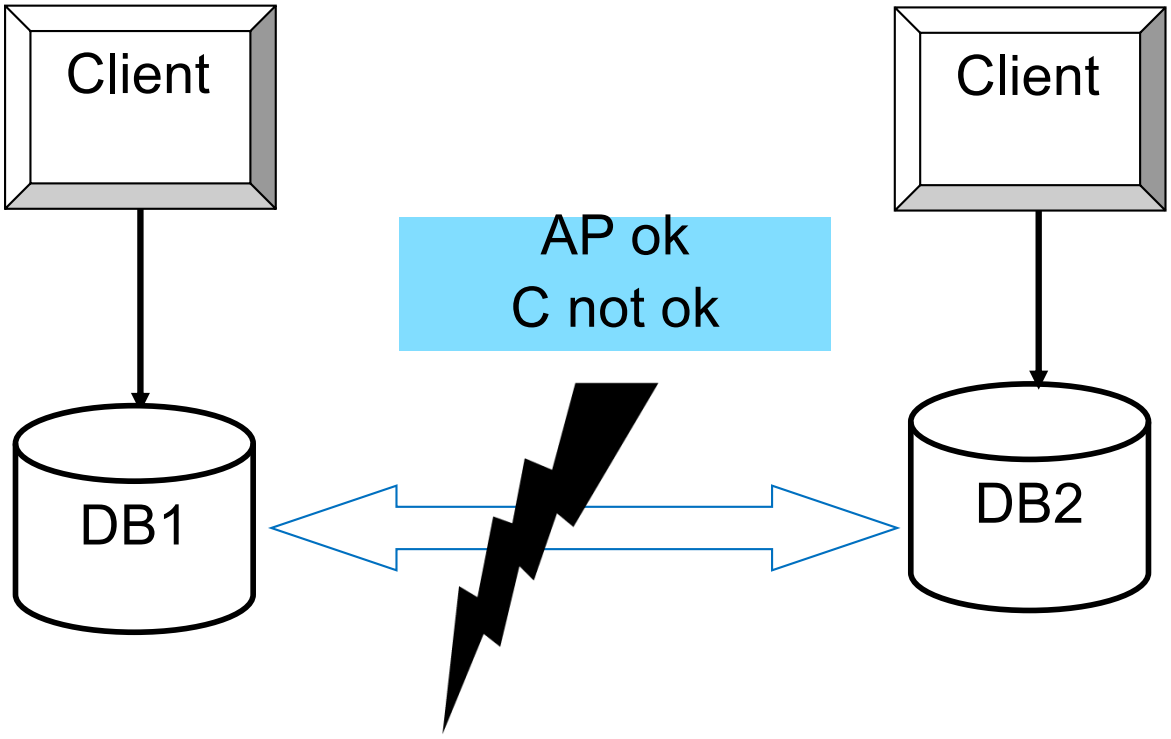
System	Data Model	Query Interface	Consistency	CAP Options	License
BigTable	Column Families	Low-Level API	Eventually Consistent	CP	Internal at Google
Google AppEng	Objects Store	Python API - GQL	Strictly Consistent	CP	Commercial
PNUTS	Key-Value Store	Low-Level API	Timeline Consistent	AP	Internal at Yahoo
Dynamo	Key-Value Store	Low-Level API	Eventually Consistent	AP	Internal at Amazon
S3	Large Objects Store	Low-Level API	Eventually Consistent	AP	Commercial
SimpleDB	Key-Value Store	Low-Level API	Eventually Consistent	AP	Commercial
RDS	Relational Store	SQL	Strictly Consistent	CA	Commercial
SQL Azure	Relational Store	SQL	Strictly Consistent	CA	Commercial
Cassandra	Column Families	Low-Level API	Eventually Consistent	AP	Open source - Apache
Hypertable	Multi-dimensional Table	Low-Level API, HQL	Eventually Consistent	AP	Open source - GNU
CouchDB	Document-Oriented Store	Low-Level API	Optimistically Consistent	AP	Open source - Apache

Strong vs Eventual Consistency

- Strong consistency (ACID)
  - All nodes see the same data values at the same time
- Eventual consistency
  - Some nodes may see different data values at the same time
  - But if we stop injecting updates, the system reaches strong consistency

- Illustration with symmetric, asynchronous replication in databases

Symmetric, Asynchronous Replication



But we have eventual consistency

– After reconnection (and resolution of update conflicts), consistency can be obtained

NoSQL (Not Only SQL): definition

- Specific “DBMS” for web-based data (Data Store)
  - Specialized data model
    - Key-value, table, document, graph, multimodal
  - Trade relational DBMS properties
    - Full SQL, ACID transactions, data independence
  - For
    - Simplicity (schema, basic API)
    - Scalability and performance
    - Flexibility for the programmer (integration with programming language)

- NB: SQL is just a language and has nothing to do with the story

NoSQL Approaches

- Characterized by the data model, in increasing order of complexity:
    1. Key-value: DynamoDB
    2. Tabular: Bigtable
    3. Document: MongoDB
    4. Graph: Neo4J
    5. Multimodel: OrientDB
- What about object DBMS or XML DBMS?
    - Were there much before NoSQL
    - Sometimes presented as NoSQL
    - But not really scalable

Key-value Stores

- Simple (key, value) data model
  - Key = unique id
  - Value = text, binary data, structured data, ...
- Simple queries
  - Put (key, value)
    - Inserts (key, value) pair
  - Value = get (key)
    - Returns value associated with key
  - {(key, value)} = get_range (key1, key2)
    - Returns data whose key is in interval [key1, key2]

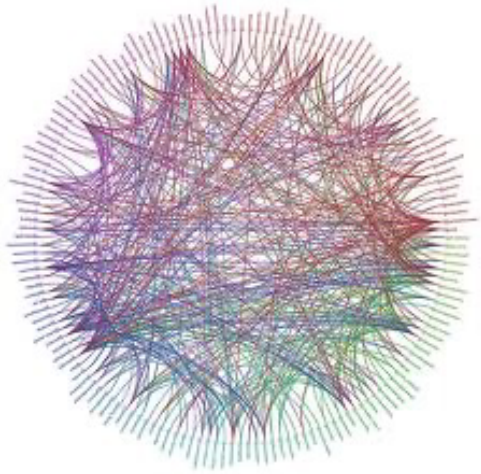
Document Stores

- Main applications: Document systems, Content Management Systems, Catalogs, Personalization, Analysis of messages (tweets, etc.) in real-time, …
- Documents:
  - Hierarchical structure, with nesting of elements
  - Weak structuring, with "similar" elements
  - Base types: text, but also integer, real, date, etc.
- Two main data models:
  - XML (eXtensible Markup Language): W3C standard (1998) for exchanging data on the Web
    - Complex and heavy
  - JSON (JavaScript Object Notation) by Douglas Crockford (2005) for exchanging data JavaScript
    - Simple and light

Tabular Stores: BigTable

- Database storage system for a shared-nothing cluster
  - Uses GFS to store structured data, with fault-tolerance and availability
- Used by popular Google applications
  - Google Earth, Google Analytics, Google+, ...
- Basis for popular Open Source implementations
  - Hadoop Hbase on top of HDFS (Apache & Yahoo)
- Specific data model that combines aspects of row-store and column-store DBMS
  - Rows with multi-valued, timestamped attributes
- Dynamic partitioning of tables for scalability

Graph DBMS



- Database graphs
  - Very big: billions of nodes and links
  - Many: millions of graphs
- Main applications
  - Social networks
    - Recommendation, sharing, sentiment analysis
  - Master data management
    - Reference business objects, data governance
  - Fraud detection in real-time
    - E-commerce, insurance, ...
  - Enterprise networks
    - Impact analysis, QoS
  - Identity management
    - Group management, provenance

Multimodel Stores: OrientDB

- Integration of key-value, document and graph
  - Extension of object model, with direct connections between objects/nodes
  - SQL extended with graph traversals
- Distributed architecture
  - Graph partitioning in a cluster
  - Symmetric replication between data centers
  - ACID transactions
  - Web technologies: JSON, REST

Main NoSQL Systems

Vendor	Product	Category	Comments
Amazon	DynamoDB	KV	Proprietary
Apache	Cassandra Accumulo	KV Tabular	Open source, Orig. Facebook Open source, Orig. NSA
Couchbase	Couchbase	KV, document	Origin: MemBase
Google	Bigtable	Tabular	Proprietary, patents
FaceBook	RocksDB	KV	Open source
Hadoop	Hbase	Tabular	Open source, Orig. Yahoo
LinkedIn	Voldemort Expresso	KV Document	Open source ACID transactions
10gen	MongoDB	Document	Open source
Oracle	NoSQL	KV	Based on BerkeleyDB
OrientDB	OrientDB	Graph, KV, document	Open source, ACID transactions
Neo4J.org	Neo4J	Graph	Open source, ACID transactions
Ubuntu	CouchDB	Document	Open source

NewSQL

- Pros NoSQL
  - Scalability
    - Often by relaxing strong consistency
  - Performance
  - Practical APIs for programming
- Pros Relational
  - Strong consistency
  - Transactions
  - Standard SQL
    - Makes it easy for tool vendors (BI, analytics, …)
- NewSQL = NoSQL/relational hybrid

Main NewSQL systems

Vendor	Product	Objective	Comment
Clustrix Inc., San Francisco	Clustrix	Analytics and transactional	First version out in 2006
CockroachDB Labs, NY	CockroachDB	Transactional	By ex-googlers. Open source inspired by F1, based on RocksDB
Google	F1/Spanner	Transactional	Proprietary
SAP	HANA	Analytics	In-memory, column-oriented
MemSQL Inc.	MemSQL	Analytics	In-memory, column/row-oriented, compatible with MySQL
LeanXcale, Madrid	LeanXcale	Analytics and transactional	Based on Apache Derby and Hbase Multistore access (KV, Hadoop, CEP, etc.)
NuoDB, Cambridge	NuoDB	Analytics and transactionall	Solution cloud (Amazon)
GitHub	TiDB	Transactional	Open source inspired by Google F1
VoltDB Inc.	VoltDB	Analytics and transactional	Open source and proprietary versions In-memory

Which Data Store for What?

Category	Systems	Requirements
Key-value	DynamoDB, SimpleDB, Cassandra	Access by key Flexibility (no schema) Very high scalability and performance
Document	MongoDB, CouchDB, Expresso	Web content management Flexibility (no schema) Limited transactions
Tabular	BigTable, Hbase, Accumulo	Very big collections Scalability and high availability
Graph	Neo4J, Sparcity, Titan	Efficient storage and management of large graphs
Multimodel	OrientDB, ArangoDB	Integrated key-value, document and graph management
NewSQL	Google F1, CockroachDB, VoltDB	ACID transactions , flexibility and scalability SQL and key-value access

Polystores

- Also called *Multistores*
- Provide integrated access to multiple cloud data stores such as NoSQL, HDFS and RDBMS
- Great for integrating structured (relational) data and big data
- Much more difficult than distributed databases
- A major area of research & development

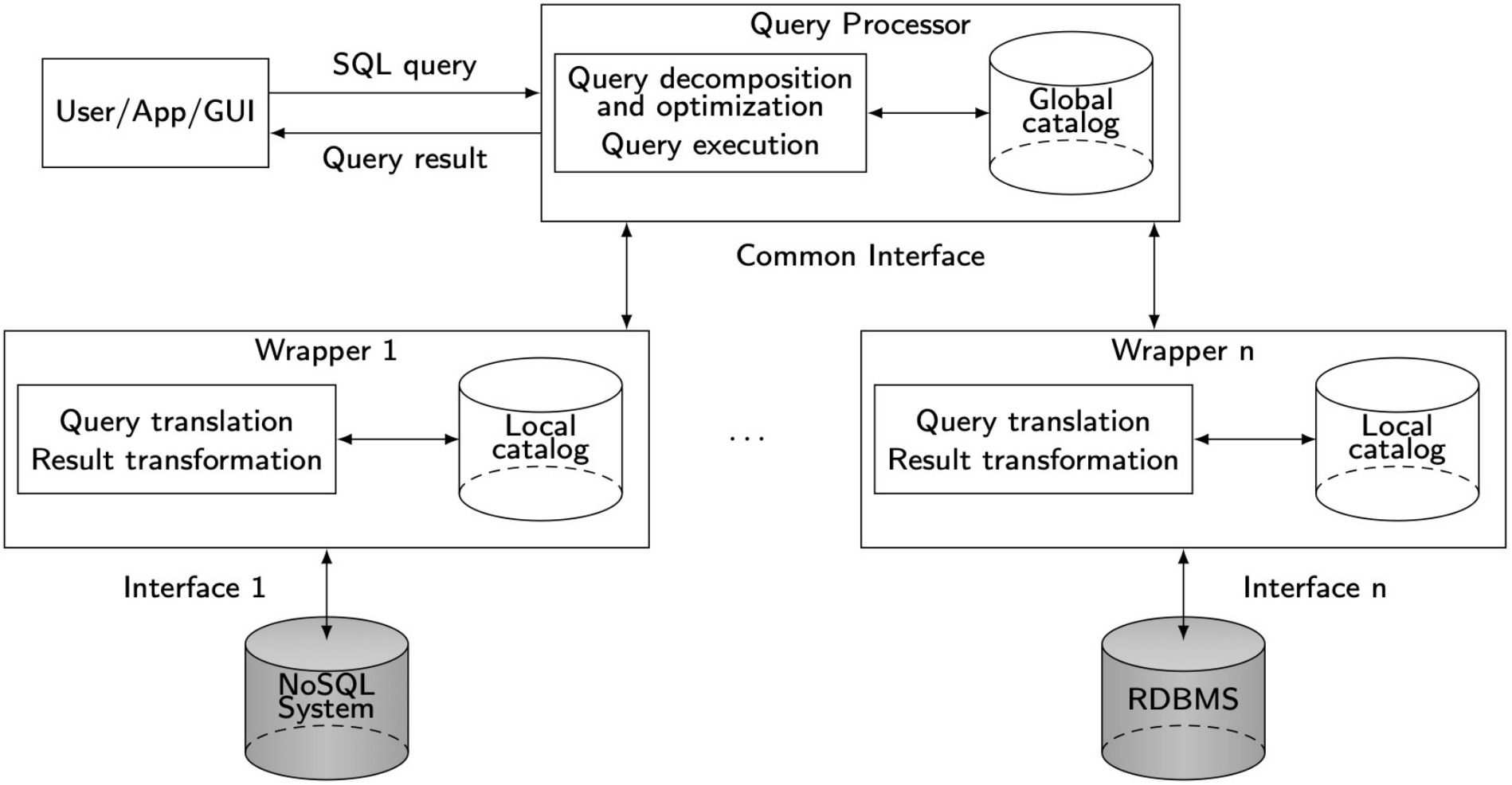
Differences with Distributed DBS

- Multidatabase systems
  - Few databases (e.g. less than 10)
    - Corporate DBs
  - Powerful queries (with updates and transactions)
- Web data integration systems
  - Many data sources (e.g. 1000's)
    - DBs or files behind a web server
  - Simple queries (read-only)
- Mediator/wrapper architecture
- In the cloud, more opportunities for efficient multistore architecture
  - No restriction to where mediator and wrapper components need be installed
- Classification of Polystores: divide polystores based on the level of coupling with the underlying data stores
  - Loosely-coupled
  - Tightly-coupled
  - Hybrid

Loosely-coupled Polystores

- Reminiscent of multidatabase systems
  - Mediator-wrapper architecture
  - Deal with autonomous data stores
  - One common interface to all data stores
  - Common interface translated to local API
- Examples
  - BigIntegrator (Uppsala University)
  - Forward (UC San Diego)
  - QoX (HP Labs)

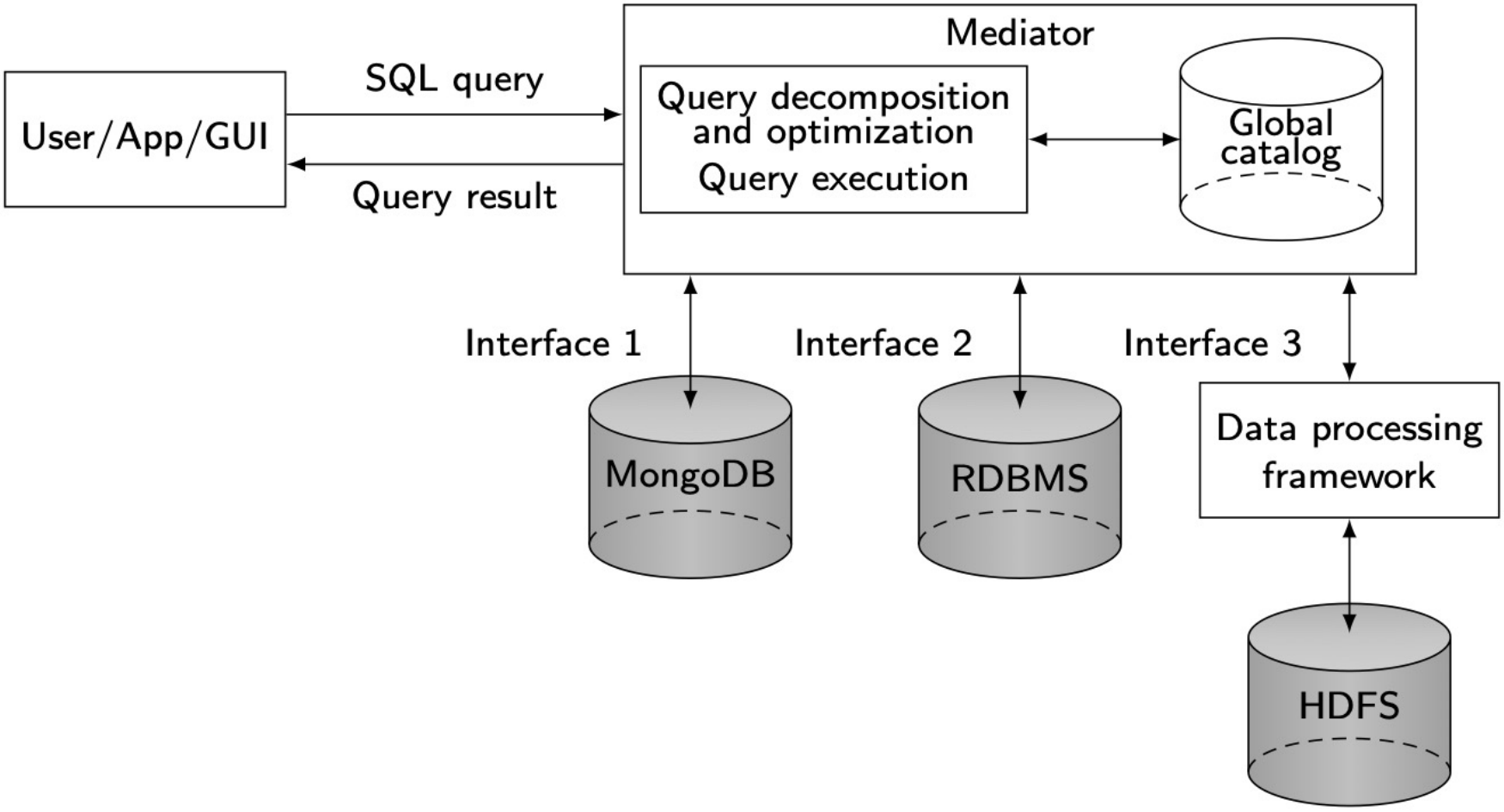
Architecture



Tightly-coupled Polystores

- Use the local interfaces of the data stores
- Use a single query language for data integration in the query processor
- Allow data movement across data stores
- Optimize queries using materialized views or indexes
- Examples
  - Polybase (Microsoft Research, Madison)
  - HadoopDB (Yale Univ. & Brown Univ.)
  - Estocada (Inria, France)

Architecture



Comparison - Functionality

Polystore	Objective	Data model	Query language	Data stores
Loosely-coupled				
BigIntegrator	Querying relational and cloud data	Relational	SQL-like	BigTable,RDBMS
Forward	Unyfing relational and NoSQL	JSON-based	SQL++	RDBMS, NoSQL
QoX	Analytic data flows	Graph	XML based	RDBMS, ETL
Tightly-coupled				
Polybase	Querying Hadoop from RDBMS	Relational	SQL	HDFS, RDBMS
HadoopDB	Querying RDBMS from Hadoop	Relational	SQL-live (HiveQL)	HDFS, RDBMS
Estocada	Self-tuning	No common model	Native QL	RDBMS, NoSQL
Hybrid				
SparkSQL	SQL on top of Spark	Nested	SQL-like	HDFS, RDBMS
BigDAWG	Unifying relational and NoSQL	No common model	Island query languages	RDBMS, NoSQL, Array DBMS, DSMSs
CloudMdsQL	Querying relational and NoSQL	JSON-based	SQL-like with native subqueries	RDBMS, NoSQL, HDFS

Comparison - Implementation

Polystore	Objective	Data model	Query language	Data stores
Loosely-coupled				
BigIntegrator	Importer,absorber, finalizer	LAV	Access filters	Heuristics
Forward	Query processor	GAV	Data store capabilities	Cost-based
QoX	Dataflow engine	No	Data/function shipping	Cost-based
Tightly-coupled				
Polybase	HDFS bridge	GAV	Query splitting	Cost-based
HadoopDB	SMS planer, db conector	GAV	Query splitting	Heuristics
Estocada	Storage advisor	Materialzed views	Query rewriting	Cost-based
Hybrid				
SparkSQL	Dataframes	Nested	In-memory caching	Cost-based
BigDAWG	Island query	GAV within islands	Function/datashipping	Heuristics
CloudMdsQL	Query planner	No	Bind join	Cost-based

Conclusions for Polystores

1. The ability to integrate relational data (stored in RDBMS) with other kinds of data stores
2. The growing importance of accessing HDFS within Hadoop
3. Most systems provide a relational/SQL-like abstraction
   - QoX has a more general graph abstraction to capture analytic dataflows
   - BigDAWG allows the data stores to be directly accessed with their native (or island) languages

Web Data Management

Vera Goebel

Department of Informatics, University of Oslo

Web Models
Web Search
Web Querying
Web Data Exchange
Web Data Integration

Web Overview (status 2011)

- Publicly indexable web:
  - More than 25 billion static HTML pages.
  - Over 53 billion pages in dynamic web
- Deep web (hidden web)
  - Over 500 billion documents
- Most Internet users gain access to the web using search engines.
- Very dynamic
  - 23% of web pages change daily.
  - 40% of commercial pages change daily.
- Static versus dynamic web pages
- Publicly indexable web (PIW) versus hidden web (or deep web)

Current status: http://www.worldwidewebsize.com/

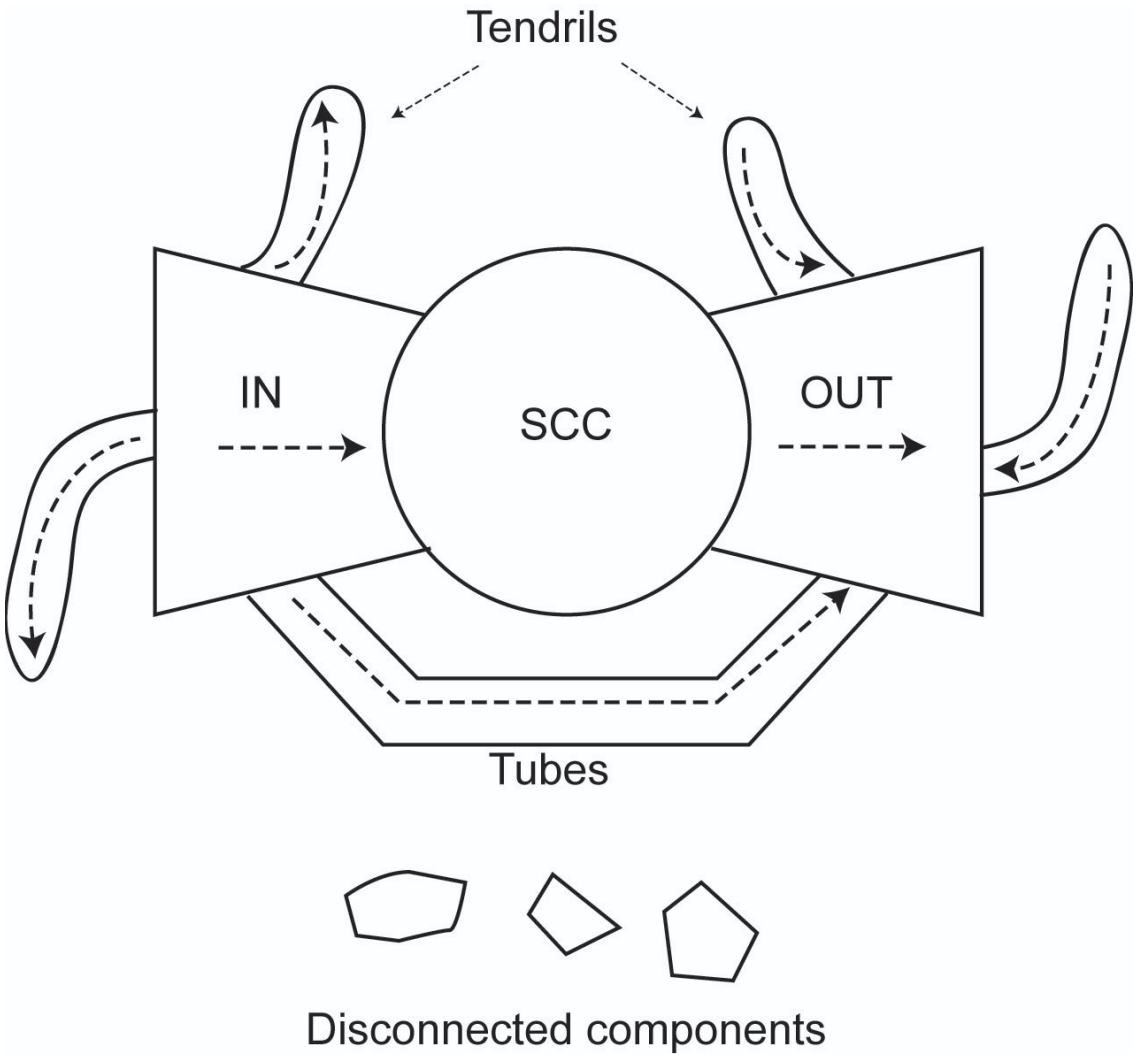
Properties of Web Data

- Lack of a schema
  - Data is at best “semi-structured”
  - Missing data, additional attributes, “similar” data but not identical
- Volatility
  - Changes frequently
  - May conform to one schema now, but not later
- Scale
  - Does it make sense to talk about a schema for Web?
  - How do you capture “everything”?
- Querying difficulty
  - What is the user language?
  - What are the primitives?
  - Search engines or metasearch engines?

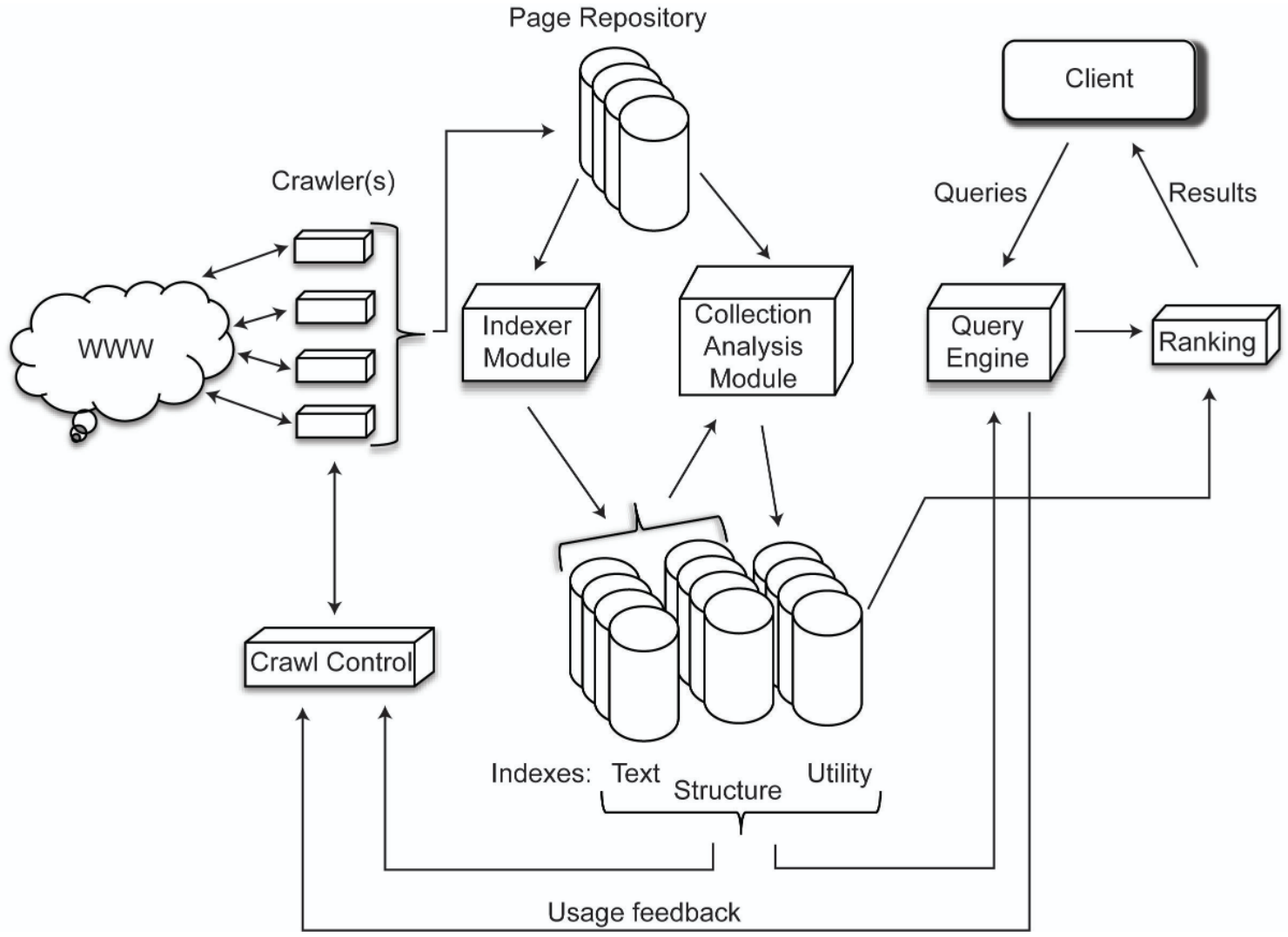
Web Model

- Web graph
  - Pages are vertices
  - Hyperlinks between pages are edges
  - Semi-structured data
- Properties
  - Volatile
  - Sparse
  - Self-organizing
  - “Small-world” graph
  - Power law graph

Structure of the Web



Search Engine Architecture



Web Crawling

- What is a crawler?
- Crawlers cannot crawl the whole Web. It should try to visit the “most important” pages first.
- Importance metrics:
  - Measure the importance of a Web page
  - Ranking
    - Static
    - Dynamic
- Ordering metric
  - How to choose the next page to crawl

Web Crawler Types

- Many Web pages change frequently
  -> crawler has to revisit already crawled pages
  → incremental crawlers

- Some search engines specialize in searching pages belonging to a particular topic
  → focused crawlers

- Search engines use multiple crawlers sitting on different machines and running in parallel
  -> coordinate parallel crawlers to prevent overlapping
  → parallel crawlers

Indexing

- Structure index
  - Link structure
- Text index
  - Indexing the content
  - Suffix arrays, inverted index, signature files
  - Inverted index most common
- Difficulties of inverted index
  - The huge size of the Web
  - The rapid change makes it hard to maintain
  - Storage versus performance efficiency

Web Querying

- Why Web Querying?
  - It is not always easy to express information requests using keywords.
  - Search engines do not make use of *Web topology* and *document structure* in queries.
- Early Web Query Approaches
  - Structured (Similar to DBMSs): Data model + Query Language
  - Semi-structured: e.g. Object Exchange Model (OEM)

Web Querying

- Question Answering (QA) Systems
  - Finding answers to natural language questions, e.g. *What is Computer?*
  - Analyze the question and try to guess what type of information that is required.
  - Not only locate relevant documents but also extract answers from them.

Approaches to Web Querying

- Search engines and metasearchers
  - Keyword-based
  - Category-based
- Semistructured data querying
- Special Web query languages
- Question-Answering

Semistructured Data Querying

- Basic principle: Consider Web as a collection of semistructured data and use those techniques
- Uses an edge-labeled graph model of data
- Example systems & languages:
  - Lore/Lorel
  - UnQL
  - StruQL

Evaluation

- Advantages
  - Simple and flexible
  - Fits the natural link structure of Web pages
- Disadvantages
  - Data model too simple (no record construct or ordered lists)
  - Graph can become very complicated
    - Aggregation and typing combined
    - DataGuides
  - No differentiation between connection between documents and subpart relationships

Web Query Languages

- Basic principle: Take into account the documents' content and internal structure and external links
- Graph structures are more complex
- First generation
  - Model the web as interconnected collection of atomic objects
  - WebSQL
  - W3QL
  - WebLog
- Second generation
  - Model the web as a linked collection of structured objects
  - WebOQL
  - StruQL

Evaluation

- Advantages
  - More powerful data model - Hypertree
    - Ordered edge-labeled tree
    - Internal and external arcs
  - Language can exploit different arc types (structure of the Web pages can be accessed)
  - Languages can construct new complex structures.
- Disadvantages
  - You still need to know the graph structure
  - Complexity issue

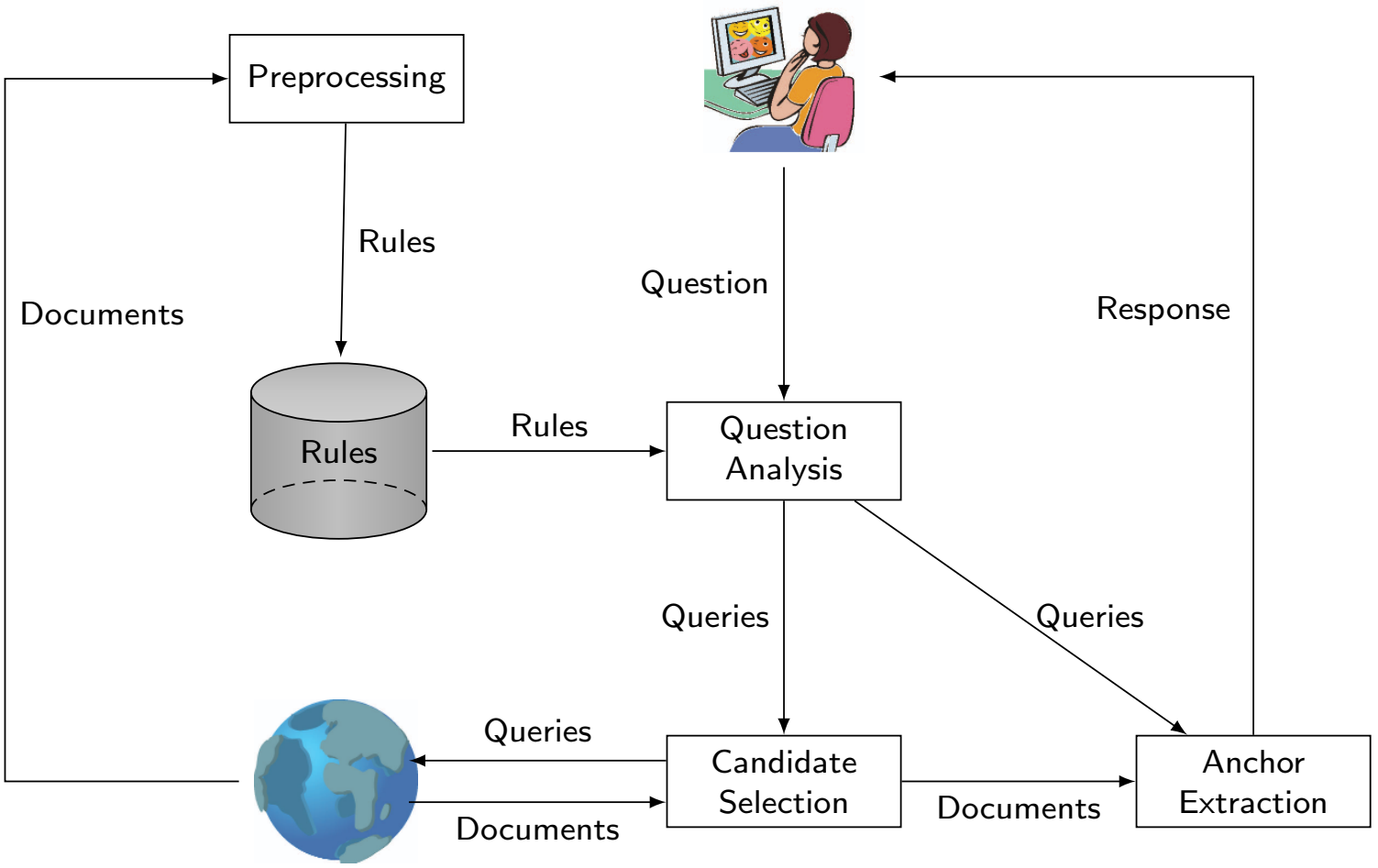
Question-Answer Approach

- Basic principle: Web pages that could contain the answer to the user query are retrieved and the answer extracted from them.
- NLP and information extraction techniques
- Used within IR in a closed corpus; extensions to Web
- Examples
  - QASM
  - Ask Jeeves
  - Mulder
  - WebQA

Question-Answer Systems

- Analyze and classify the question, depending on the expected answer type.
- Using IR techniques, retrieve documents which are expected to contain the answer to the question.
- Analyze the retrieved documents and decide on the answer.

Question-Answer System



Searching The Hidden Web

- Publicly indexable web (PIW) vs. hidden web
- Why is Hidden Web important?
  - Size: huge amount of data
  - Data quality
- Challenges:
  - Ordinary crawlers cannot be used.
  - The data in hidden databases can only be accessed through a search interface.
  - Usually, the underlying structure of the database is unknown.

Searching The Hidden Web

- Crawling the Hidden Web
  - Submit queries to the search interface of the database
    - By analyzing the search interface, trying to fill in the fields for all possible values from a repository
    - By using agents that find search forms, learn to fill them, and retrieve the result pages
  - Analyze the returned result pages
    - Determine whether they contain results or not
    - Use templates to extract information

Searching The Hidden Web

- Metasearching
  - Database selection – Query Translation – Result Merging
  - Database selection is based on *Content Summaries*.
  - Content Summary Extraction:
    - RS-Ord and RS-Lrd
    - Focused Probing with Database Categorization

Web Data Exchange

- XML basics

Why Use XML with a DBS?

Early solutions:

- XML as transport format
  - to/from database
- XML for semistructured data
  - schema to be stored not known at design time
- XML document archiving
  - retaining complete XML documents
- Application domain -> choice of XML-DBS

XML – In a Nutshell

Extensible Markup Language, World Wide Web Consortium (W3C)

- Standard of the W3C in 1998
- Documents (content)
- XML Schema or DTD (structure)
- Namespaces
- XSL, XSLT (stylesheets, transformations)
- XPath, XPointer, XLink
- XQuery

- Tagged elements with attributes
  - Extensible, self-describing
- Tag names must be unique (use namespaces)

XML Overview

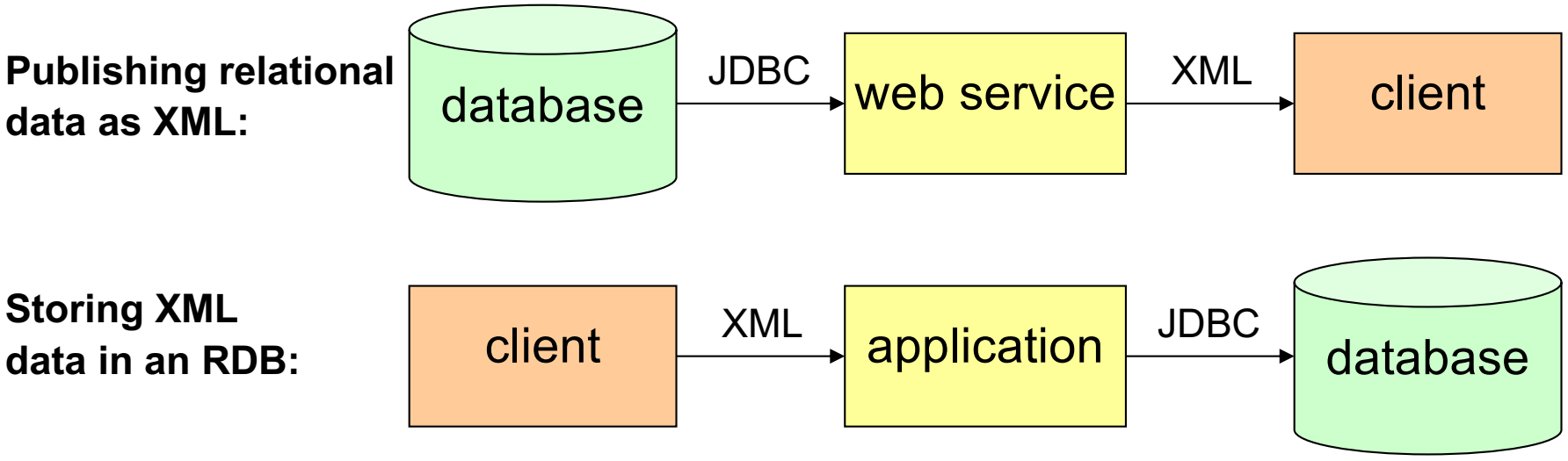
- Data is divided into pieces called elements
- Elements can be nested, but not overlapped
  - Nesting represents hierarchical relationships between the elements
- Elements have attributes
- Elements can also have relationships to other elements (ID-IDREF)
- Can be represented as a graph, but usually simplified as a tree
  - Root element
  - Zero or more child elements representing nested subelements
  - Document order over elements
  - Attributes are also shown as nodes

XML Document

```
<? xml version="1.0" ?>
<racingTeams xmlns="http://www.cart.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
    xsi:schemaLocation="http:// www.cart.org team.xsd">
    <team>
        <name>"Shell</name>
        <car number=3><make>"Helix Ford"</make>
            <engine id="47456"/></car>
        <car number=4><make>"Helix Ford"</make>…</car>
        <driver><name>"Steve Johnson"</name>
            <points>2217</points></driver >
        <driver><name>"Paul Radisich"</name></driver >
        <spares><engine id="102555"/>
            <engine id="102556"/></spares>
    </team>
    . . .
</racingTeams>
<!-- Each document has one root element (e.g., racingTeams) whose name
    matches a schema element or <!DOCTYPE name> -->
```

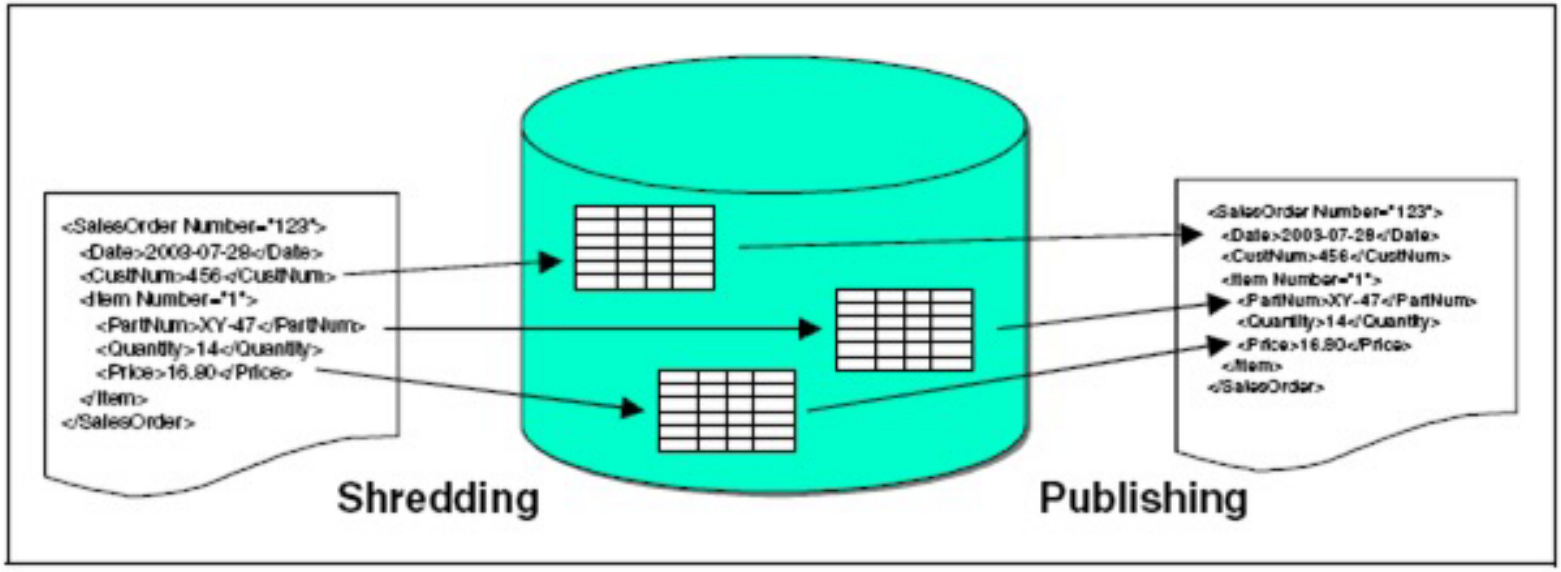
XML as Transport Format

- When
  - publishing as XML data stored in a DB, or storing data arriving as XML documents
    - between an existing DB and applications
    - between two DBs
- Why XML
  - platform independence



XML-Enabled DBS

- For data-centric XML documents
- When XML is used as a data transport format
- How
  - Store by shredding
  - Publish by composing



Properties of XML-Enabled DBS

- DB schema models primarily the *data* in the XML document
  - Data model is ”traditional”
    - relational/object-relational/...
  - No XML ”visible” in DB
- Keeps existing data and applications intact
  - adding XML functionality is adding and configuring data transfer (data mapping) software

Properties of XML-Enabled DBS

- XML mapping software
  - integrated into the DB engine or external to it
- Lossy modelling
  - XML document as such is discarded after shredding
  - In general, XML mapping software can handle only a subclass of XML documents
    - retains information that is important to the DB model:

      data itself

      hierarchical relationships (parent, child, siblings)

      not entity references, CDATA sections, comments, processing instructions, DTD, maybe not ordering of siblings,...
  - Cannot reconstruct XML document in full detail

XML for Semistructured Data

- When
  - application domains where
    - all data formats cannot be predicted in advance +
    - large volumes of data
- Why
  - why XML: extensibility
  - why DB: need to store, manage, and query data

XML Document Archiving

- When
  - retaining XML documents
- Why
  - processing
  - contracting/legislation
  - granularity: XML document or document fragment is the logical unit

NXDBS: Native XML DBS

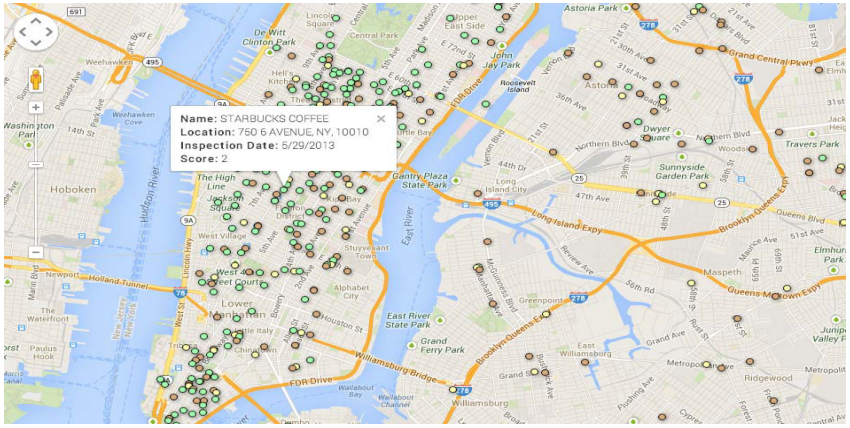
- For document-centric XML documents
  - need to be able to retain XML documents as-is
- For semi-structured data
  - when other DB models are inappropriate, e.g. since they require changing the DB schema when the XML schema evolves
- Specialized for storing XML data
  - Stores all components of the XML model intact
- XML document is the logical unit
  - (Cf. the logical unit in an RDBMS, which is a row)
- *Schema-independent NXDBS*
  - Can store any XML document regardless of its schema

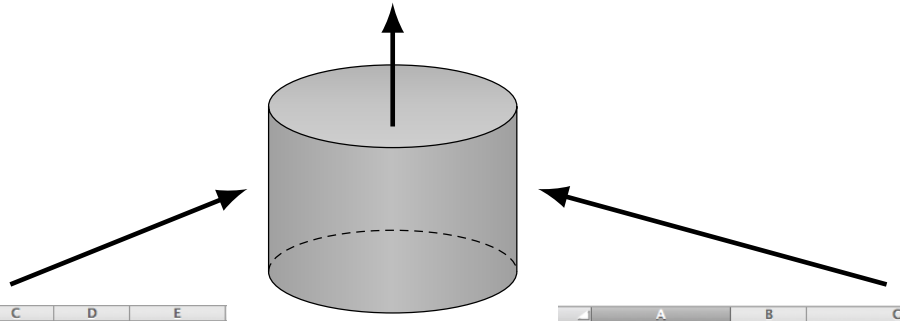
Web Data Integration

Where do we stand today? Modern solutions:

- "Big data" characteristics play a role
  - No or highly variable (among sources) schema
  - Numbers are much higher
  - Data quality is a major issue
- "Pay-as-you-go" integration
  - Data spaces → data lakes
- Some approaches
  - Web Tables / Fusion Tables
  - Semantic web & Linked Open Data (LOD)

Web Tables / Fusion Tables





	A	B	C	D	E
1	camis	inspection_d	action_code	score	current_grade
2	40363098	1/8/2013	D	11	A
3	40367790	5/1/2013	U	7	A
4	40367829	6/11/2013	D	13	A
5	40369535	4/4/2013	D	7	A
6	40369724	7/29/2013	F	28	C
7	40369782	7/1/2013	D	10	A
8	40369915	6/17/2013	U	11	A
9	40370146	11/16/2012	D	5	A
10	40370356	3/12/2013	D	6	A
11	40370781	4/30/2013	D	2	A
12	40371529	12/3/2012	D	9	A
13	40372196	4/30/2013	D	2	A
14	40372844	2/11/2013	D	11	A

	A	B	C	D	E	F
1	name	building	street	zipcode	phone	camis
2	DUNKIN' DONUTS	56	COURT STREET	11201	##########	40363098
3	MCDONALD'S	395	FLATBUSH AVENUE EXTENSION	11201	##########	40369535
4	MCDONALD'S	3267	RICHMOND AVENUE	10312	##########	40370356
5	MCDONALD'S	5804	CLARENDON ROAD	11203	##########	40373909
6	MCDONALD'S	75-50	101 AVENUE	11416	##########	40367829
7	MCDONALD'S	2154	HYLAN BOULEVARD	10306	##########	40371529
8	MCDONALD'S	3660	EAST TREMONT AVENUE	10465	##########	40372196
9	MCDONALD'S	943	FLATBUSH AVENUE	11226	##########	40367790
10	MCDONALD'S	221-28	HORACE HARDING EX	11364	##########	40372844
11	MCDONALD'S	701	UTICA AVENUE	11203	##########	40369782
12	MCDONALD'S	160-11	WILLETS POINT BOULEVARD	11357	##########	40370781
13	MCDONALD'S	106-15	71 AVENUE	11375	##########	40369724
14	MCDONALD'S	Feb-70	COPPER AVENUE	11385	##########	40377294
15	MCDONALD'S	5765-81	BROADWAY	10463	##########	40369915
16	DUNKIN' DONUTS/BASK	153-67	HORACE HARDING EXPRESSWA	11367	##########	40378035
17	MCDONALD'S	427	10 AVENUE	10001	##########	40370146

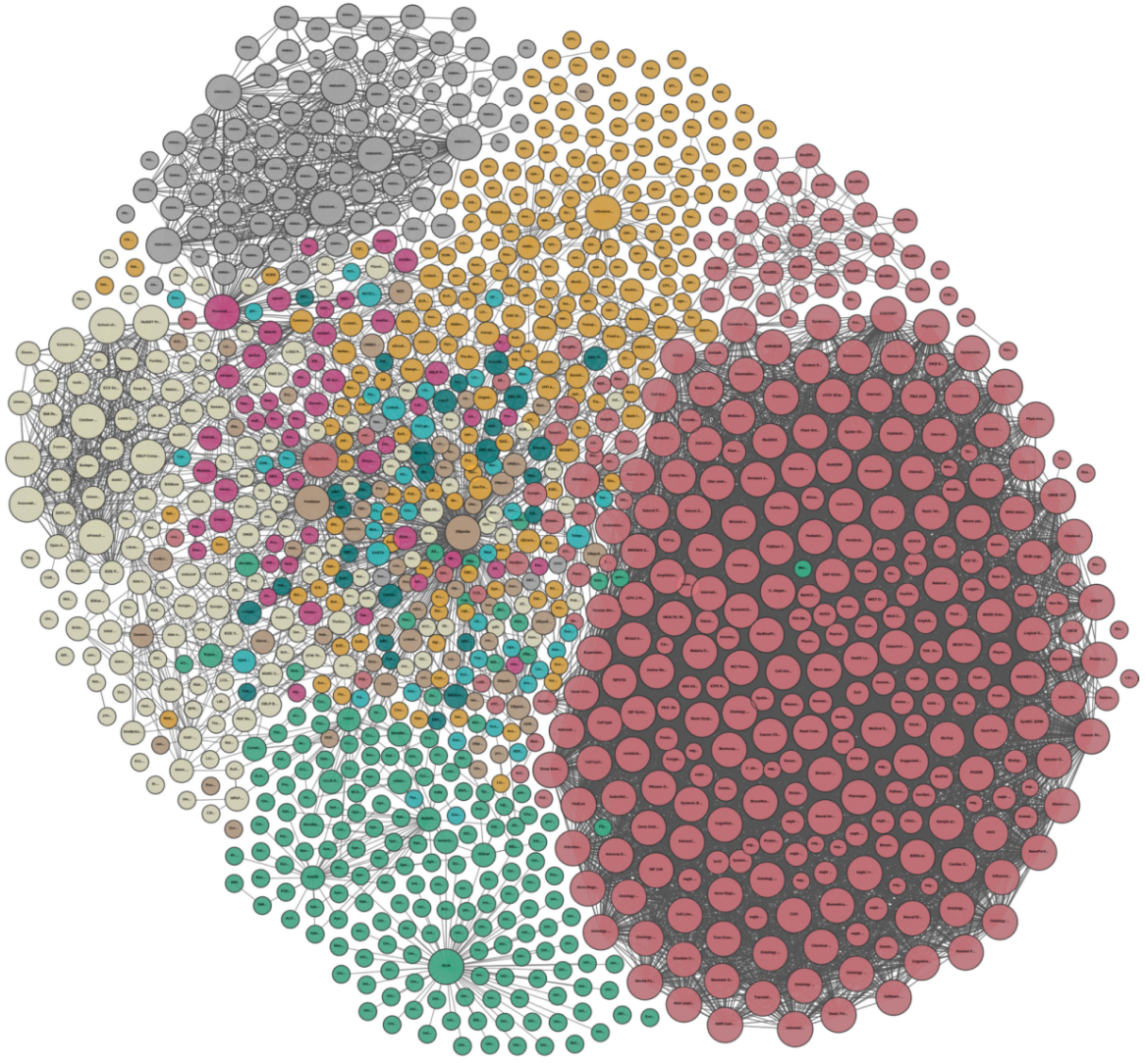
Semantic Web

- Vision: Move data from machine processable to machine understandable by integrating structured and unstructured web data and marking it up semantically
- Components:
  - Web data marked up so metadata is captured as annotations
  - Ontologies to make different data collections understandable
  - Logic-based technologies to access both metadata & ontologies

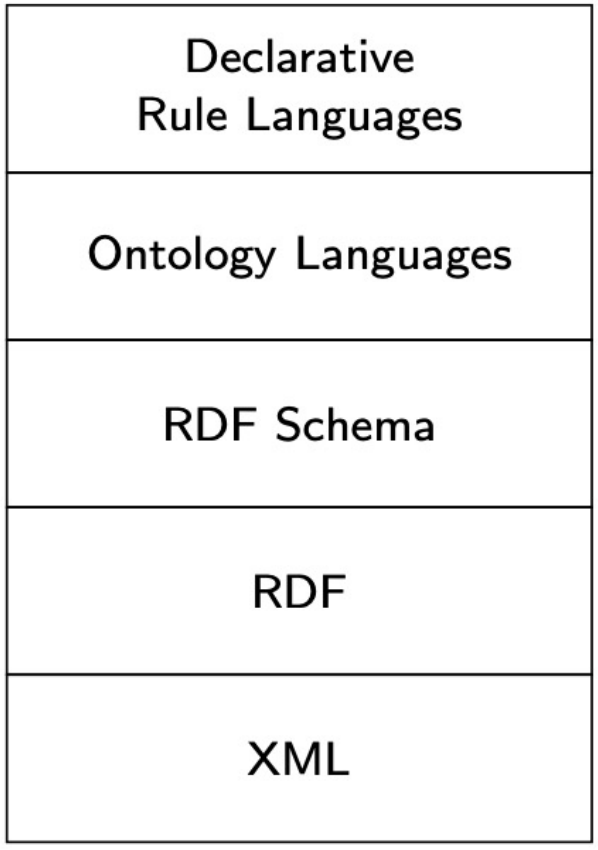
Linked Open Data (LOD)

- LOD is a realization and clarification of this vision → linkages among data is an important part of the vision
- Integrate data on the web based on four principles:
  1) All web sources (data) are locally identified by their URIs that serve as names
  2) These names are accessible by HTTP
  3) Information about data sources/entities are encoded as RDF (Resource Description Framework)
  4) Connections among datasets are established by data links

LOD Graph (status 2018)



Semantic Web Technologies



RDF: Resource Description Framework

- Data model on top of XML
- Models each fact as a set of triples
  - (subject, property (or predicate), object): ⟨s,p,o⟩
  - Subject: the entity that is described (URI or blank node)
  - Predicate: a feature of the entity (URI)
  - Object: value of the feature (URI, blank node or literal)
- Formally: an RDF triple

$$\langle s, p, o \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$$

  where
    - $\mathcal{U}$ denotes the set of URIs
    - $\mathcal{B}$ denotes the set of blank nodes
    - $\mathcal{L}$ denotes the set of literals
  - Set of RDF triples form a RDF dataset

RDF Schema (RDFS)

- Annotation of RDF data with semantic metadata
- Allows the definition of classes and class hierarchies
- Enables reasoning over RDF data (entailment)
- Built-in class definitions

RDF Query Language – SPARQL

- SPARQL Protocol and RDF Query Language
- Formally: Let $\mathcal{U}$, $\mathcal{B}$, $\mathcal{L}$, $\mathcal{V}$ denote set of all URIs, blank nodes, literals, and variables
  1) A triple pattern $(\mathcal{U} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})$ is a SPARQL expression
  2) If `P` is a SPARQL expression, then

     ```
     P FILTER R
     ```

     is also a SPARQL expression
  3) If `P1` & `P2` are SPARQL expressions, then

     ```
     P1 AND|OPT|OR P2
     ```

     are also SPARQL expressions

     2) and 3) are optional
- Basic graph pattern (BGP): set of triple patterns

RDF Data Management Approaches

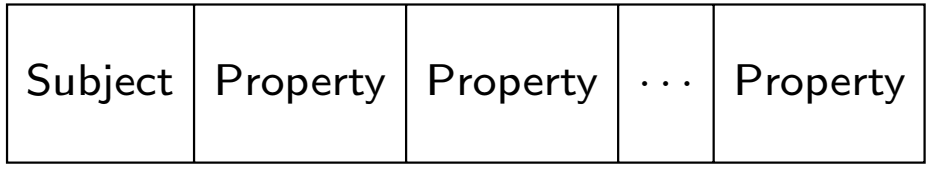
- Direct relational mapping
- Relational schema with extensive indexing
- Denormalize triples table into clustered properties
- Column-store organization
- Native graph representation

Single Table Exhaustive Indexing

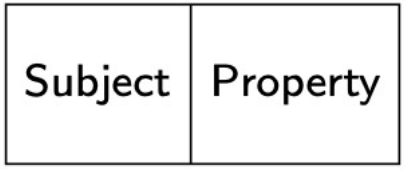
- Maintain a single table of RDF triples
- Create indexes for permutations of the three columns: SPO, SOP, PSO, POS, OPS, OSP
- RDF-3X and Hexastore
- Query processing
  - Each triple pattern can be evaluated by a range query
  - Joins between triple patterns computed using merge join
  - Join order is easy due to extensive indexing
- Advantages
  - Eliminates some of the joins { they become range queries
  - Merge join is easy and fast
- Disadvantages
  - Updates to indexes are expensive
  - Space usage

Property Tables

- Grouping by entities
- Jena
- Clustered property table: group together the properties that tend to occur in the same (or similar) subjects



Single-valued properties



Multi-valued properties

- Property class table: cluster the subjects with the same type of property into one property table

Property Tables

- DB2-RDF same approach but different structure
- Direct primary hash (DPH): organize by each subject with fixed number of (property, value) with spill
  - Properties in a given position might be different in different rows

Subject	Spill	$Prop_1$	val_1	$Prop_2$	val_2	$\cdots$	$Prop_k$	val_k

- Direct secondary hash (DSH): for multi-valued properties
  - DPH property value field has a pointer to DSH entry (*l_id*)

l_id	value

Property Tables Evaluation

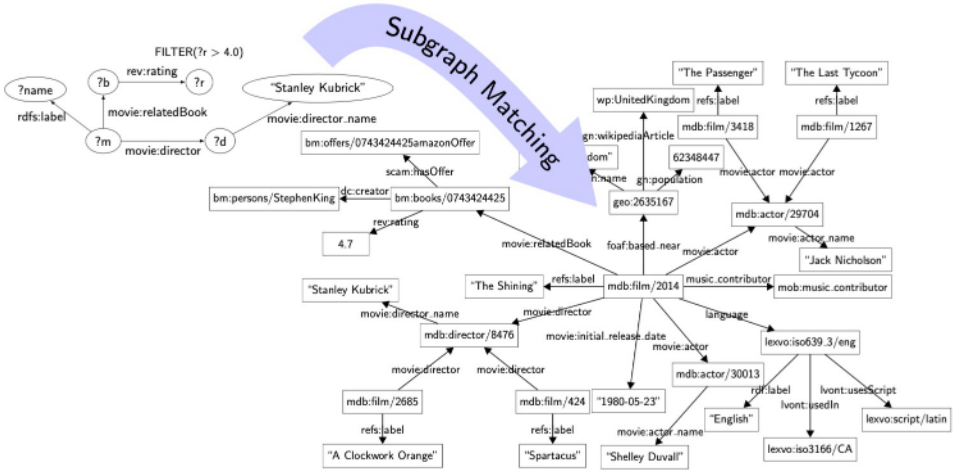
- Advantages
  - Star queries (subject-subject joins) become single table scans → fewer joins
- Disadvantages
  - Can have significant number of null values
  - Dealing with multi-valued properties requires special care
  - Is not very helpful for non-star queries
  - Clustering “similar” properties is non-trivial

Binary Tables

- Grouping by properties: For each property, build a two-column table, containing both subject and object, ordered by subjects
- *n* two-column tables
- Advantages
  - Advantages of column-oriented organization
  - Avoids null values
  - No need for (manual or automatic) clustering
  - Subject-subject joins by merge join
- Disadvantages
  - Insertions are expensive
  - More joins are required
  - Other join types do not benefit

Graph-based Processing

- Maintain graph structure of RDF data, convert SPARQL query to a query graph
- Answering SPARQL query ≡ subgraph matching using homomorphism
- gStore



- Advantages
  - Native representation → maintains the original graph structure
  - No restrictions on SPARQL queries
- Disadvantage
  - Subgraph matching is expensive

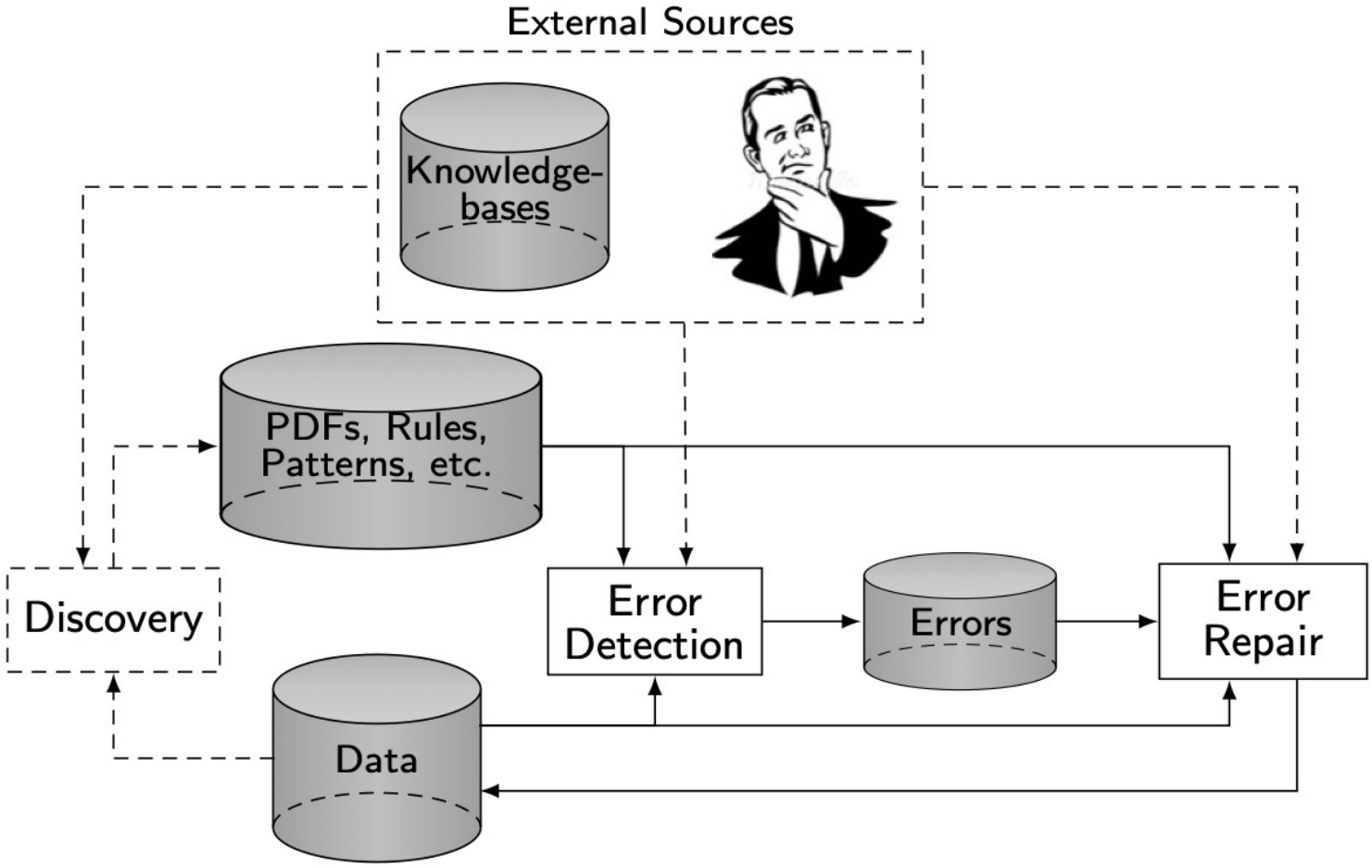
Federated RDF Systems

- Similar concerns to the relational counterparts
- Similar approaches
- Not all RDF data storage sites can execute SPARQL queries
  - SPARQL endpoints: storage sites that can execute queries
- Precompute metadata at endpoints → specify the capabilities
- Based on precomputation decompose a query and execute

Data Quality Issues

- More serious in web data integration
  - Data from far more sources
  - Data sources are not as well controlled
  - Lack of schema information in web data
    - Might have to use data-based techniques and not schema-based
- Data quality has two components
  - Data consistency
  - Veracity: authenticity and conformity of data with reality
    - Major challenge, but
    - High redundancy due to overlaps among data sources

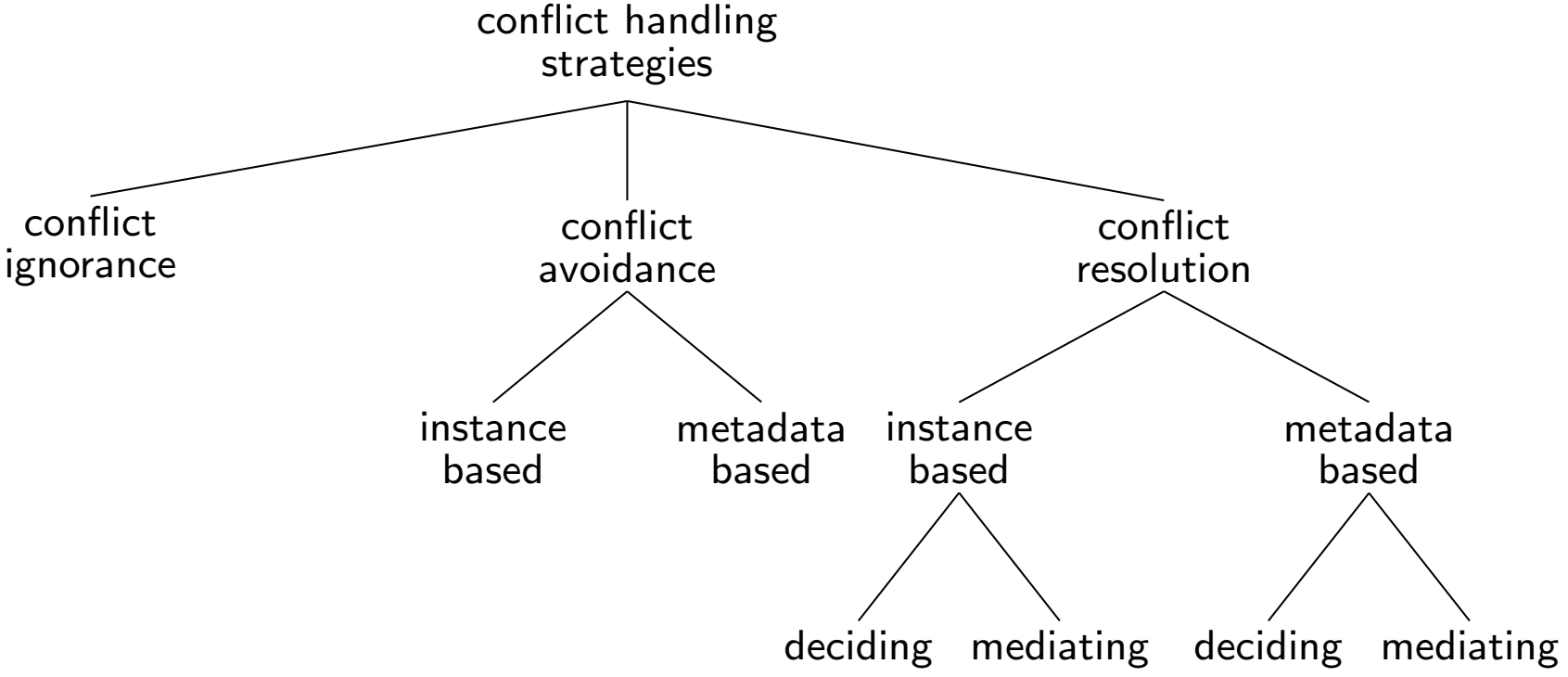
Cleaning Structured Web Data



Web Data Fusion

- Deciding the correct value for a data item that has differences among web sources
- Conflicts:
  - Uncertainty: one source has a non-null value, others have null value
    - Due to missing information (hence null values)
  - Contradiction: two or more non-null values of the same data item do not agree
- Typical enterprise data integration cleaning techniques may or may not work

Web Data Fusion Approaches



Web Source Quality

- Not all sources are equal
  - Cleaning techniques cannot assume all values to be of the same accuracy
- Web sources may copy from each other
  - Cannot ignore these dependencies
- Values can evolve over time
  - Incorrect value and outdated value are not the same thing

IN5040/9040 - H2025
Advanced Database Systems for Big Data

Vera Goebel

Thomas Plagemann

Fall 2025

Course Purpose

- The course gives an overview of new developments within data management technology
  - Emphasis on usage and applicability
  - Concepts and design, not so much about concrete systems

Organization of the Course

- Course mode:
  - Lectures: Wednesday, 3 hours, 12:15-15:00, Lille Aud. (old Ifi)
  - 10 weeks lecturing
  - Mandatory exercise (2 parts)
- Information & lecturing material:
  https://www.uio.no/studier/emner/matnat/ifi/IN5040/h25/index.html
- PhD students (IN9040):
  mandatory 45 min. Presentation
- Examination:
  - Oral (dates and modalities to be announced later)

Syllabus (Pensum)

- All slides and handouts
- Selected articles (on the Web page)
- Hueske, F., Kalavri, V.: Stream Processing with Apache Flink, O´Reilly, 2019

DBS basic knowledge & selected chapters for IN5040:

- Elmasri & Navathe: Fundamentals of Database Systems, 5th or 6th edition, Addison-Wesley
- Garcia-Molina, Ullman & Widom: Database Systems – The Complete Book, 2nd edition, Prentice Hall, 2009

Mandatory Exercise

- **Organization:** 2 parts = 2 deliveries
  each student works alone
- **Delivery date:** Hard Deadlines!
  Part 1: September 24, 2025, 23:59h
  Part 2: October 16, 2025, 23:59h
- Only students that have passed the mandatory exercise (both parts!) are allowed to take the examination!

Content Overview

- Data Stream Processing (Part 1)
- Data Stream Processing (Part 2)
- Distributed Database Systems
- Heterogeneous Multi-Database Systems
- Web & XML Data Management
- Knowledge Discovery & Data Warehouses
- Machine Learning in Medicine *
- Scalable & Cloud Data Management
- Performance Analysis & Large DBS *

*Guest Lectures