VLDB 2022

Polyglot Data Management: State of the Art & Open Challenges

Felix Kiehn
felix.kiehn@uni-hamburg.de

Mareike Schmidt
mareike.schmidt-3@uni-hamburg.de

Daniel Glake
daniel.glake@uni-hamburg.de

Fabian Panse
fabian.panse@uni-hamburg.de

Wolfram Wingerath
wolle@uni-oldenburg.de

Benjamin Wollmer
benjamin.wollmer@uni-hamburg.de

Martin Poppinga
martin.poppinga@uni-hamburg.de

Norbert Ritter
norbert.ritter@uni-hamburg.de

Who We Are

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Felix Kiehn

Martin Poppinga

HAW HAMBURG

Daniel Glake

Mareike Schmidt

Carl von Ossietzky Universität Oldenburg

Wolfram Wingerath

Fabian Panse

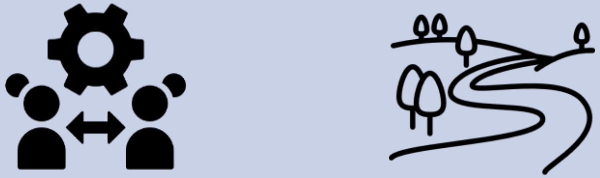
Norbert Ritter

Baqend

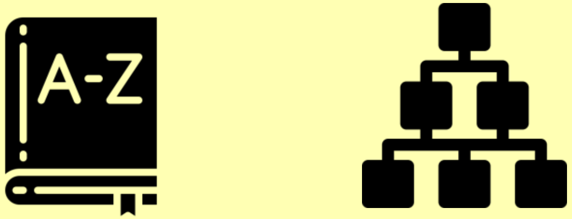
Benjamin Wollmer

Outlook

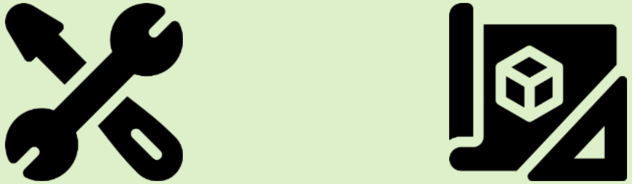
PART I:
Motivation &
Database Landscape



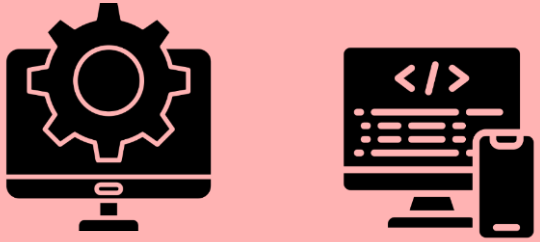
PART II:
Terminology & Taxonomies



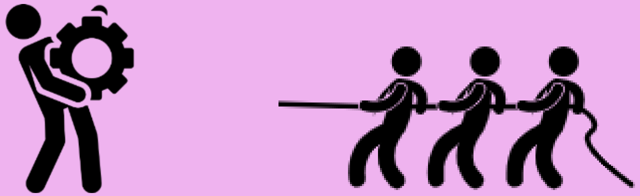
PART III:
Basic Techniques
& Concepts



PART IV:
Current Systems



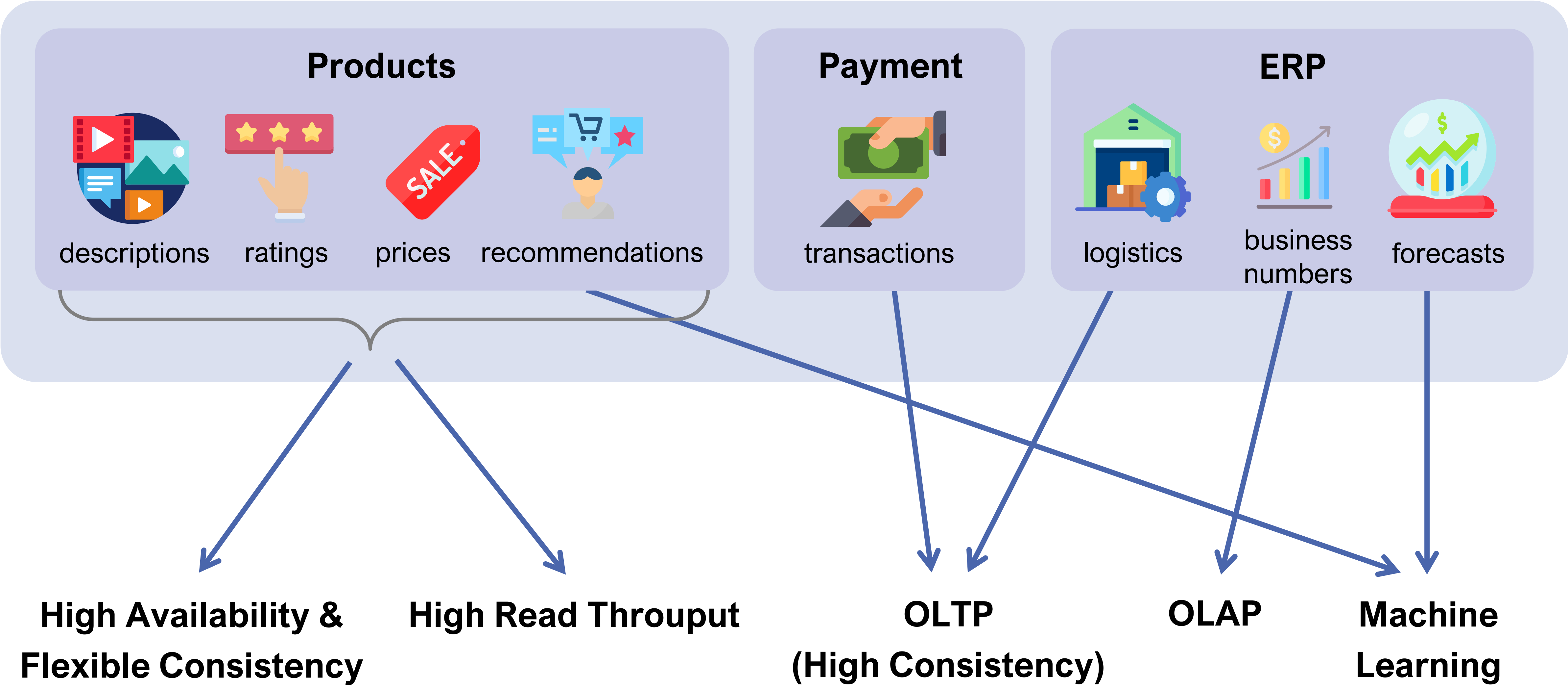
PART V:
Open Challenges



Part I

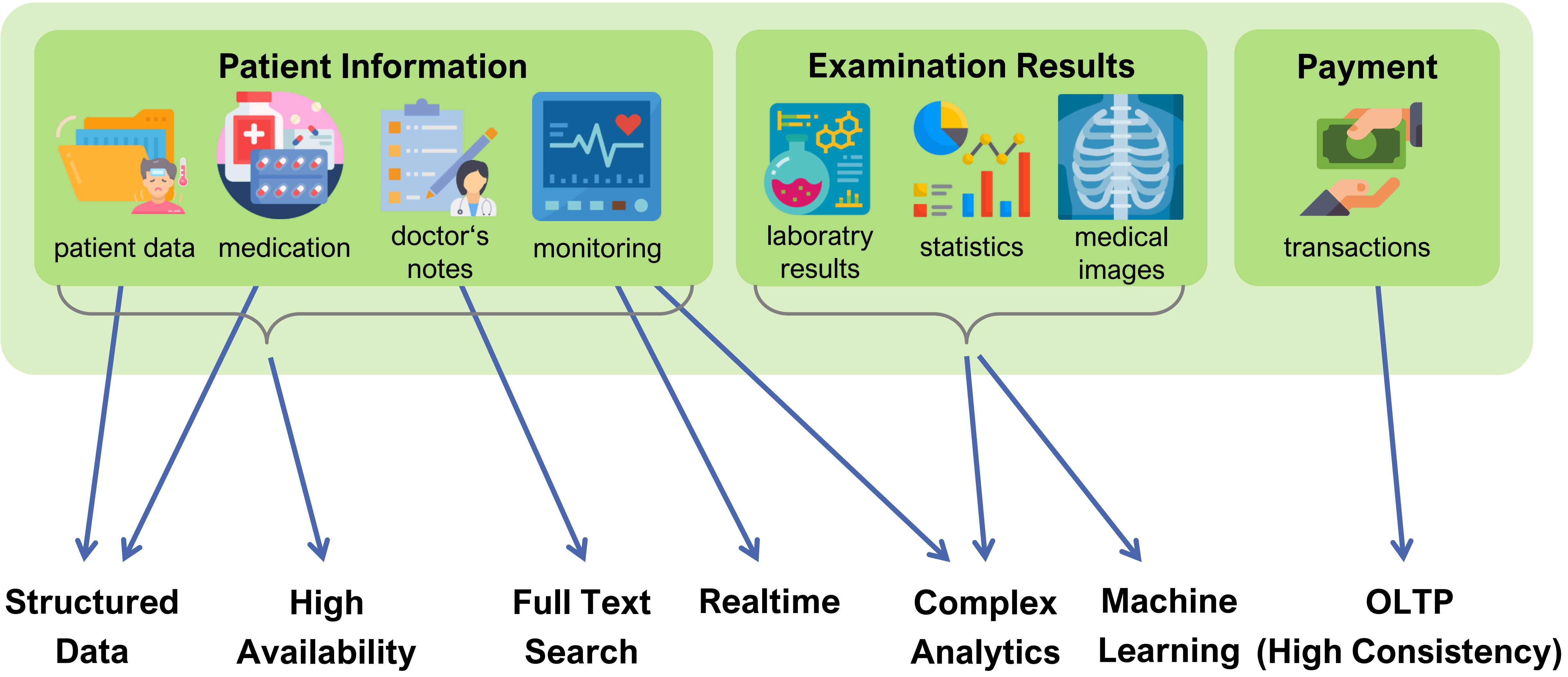
Motivation & Database Landscape

Use Case 1: E-Commerce



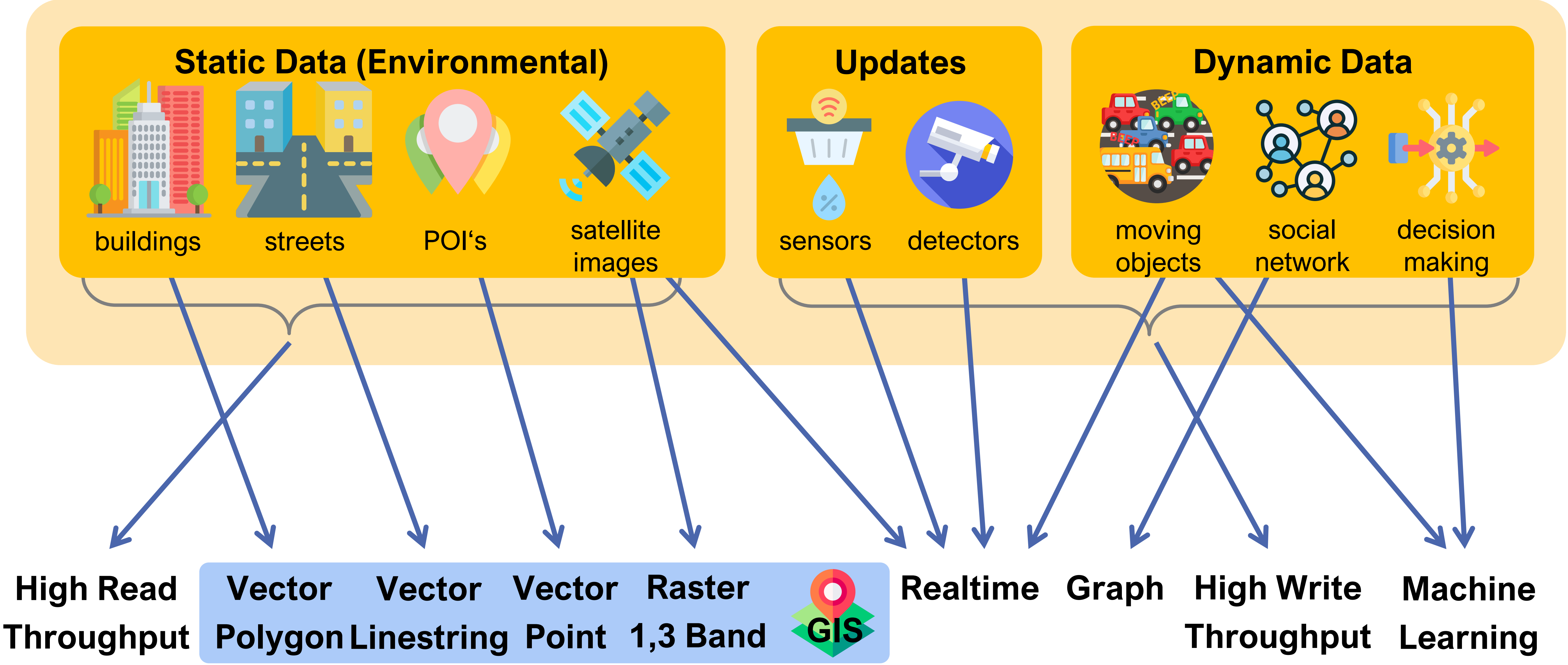
https://www.baqend.com/publications

Use Case 2: Medical Application Data (e.g. MIMIC)



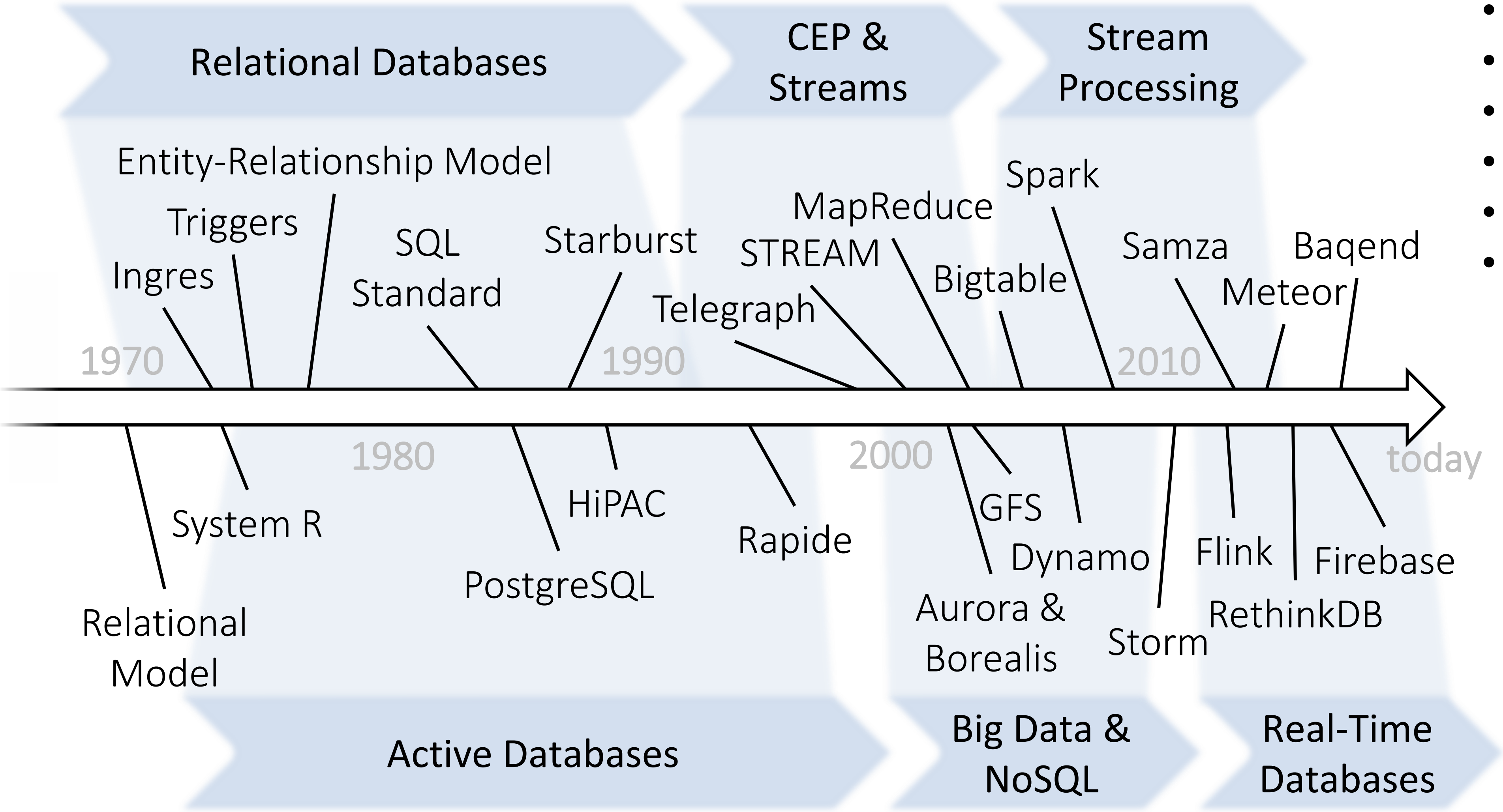
https://physionet.org/content/mimiciii

Use Case 3: Digital Twin (e.g. MARS)



https://mars.haw-hamburg.de

A Short History of Data Management



Not included:

- Timeseries DBs
- (Geo-)spatial DBs
- Object-oriented DBs
- Probabilistic DBs
- Graph Stores
- …

Typical Classification Schemes

Not included:

- Functional/non-functional properties
- Cloud vs. on-premise
- ACID vs. BASE
- …

Data Model

- Relational
- Key-Value
- Wide-Column
- Document
- Graph

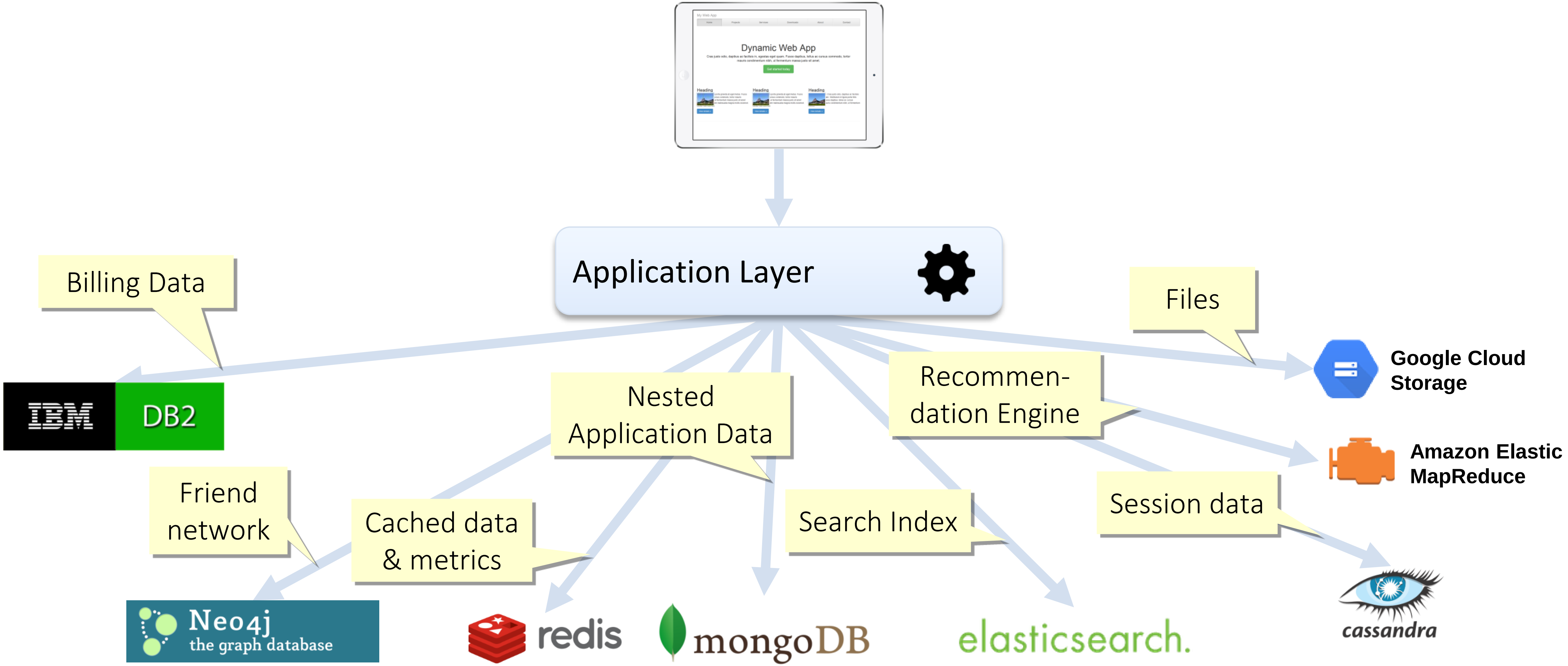
CAP Classes

- **AP**: Available & Partition Tolerant
- **CP**: Consistent & Partition Tolerant
- **CA**: Not Partition Tolerant

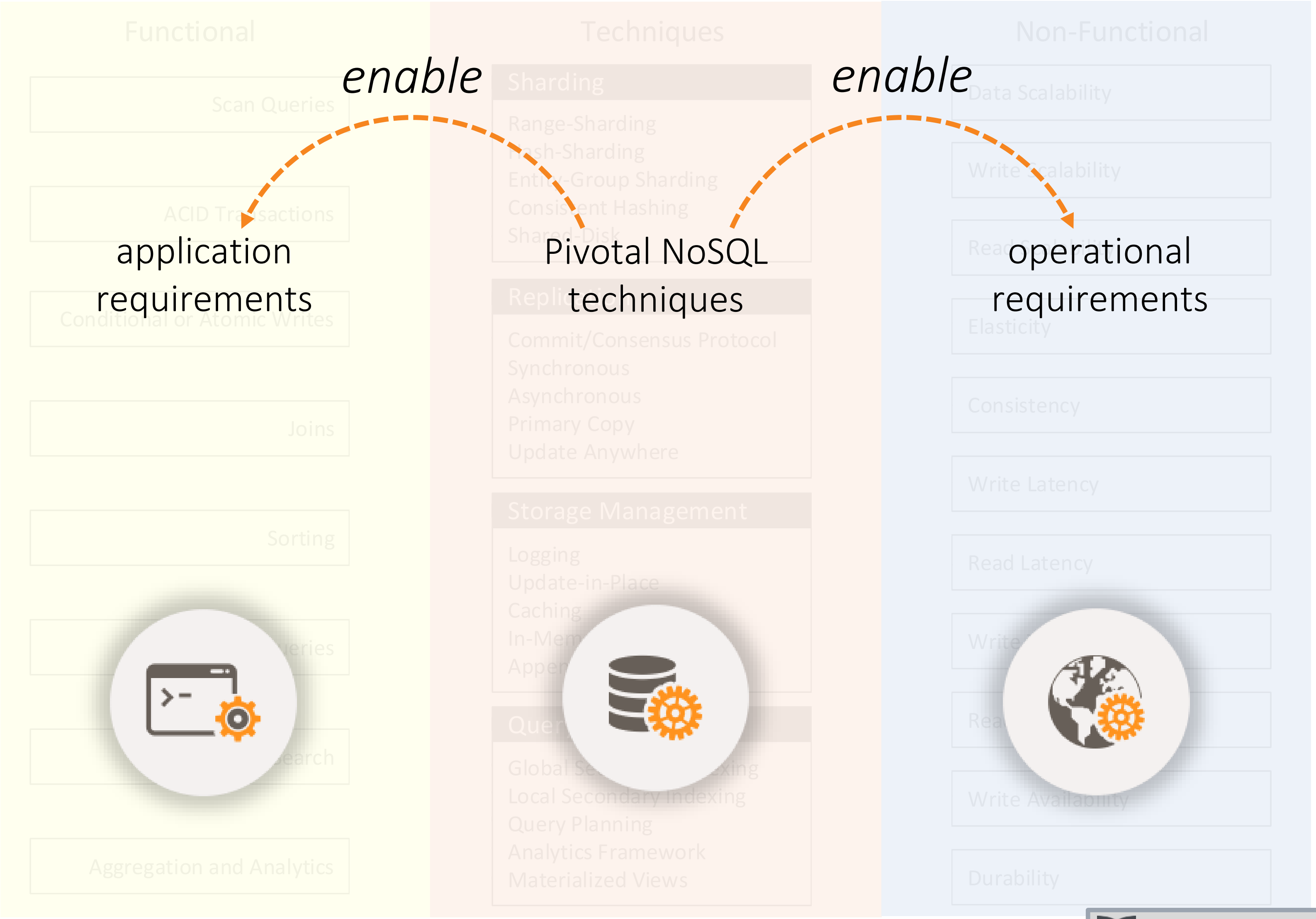
Pull vs. Push

- **Databases**: static collections
- **Real-Time DBs**: dynamic collections
- **Stream Processing**: dynamic streams

How to Choose The „Right“ Database System?

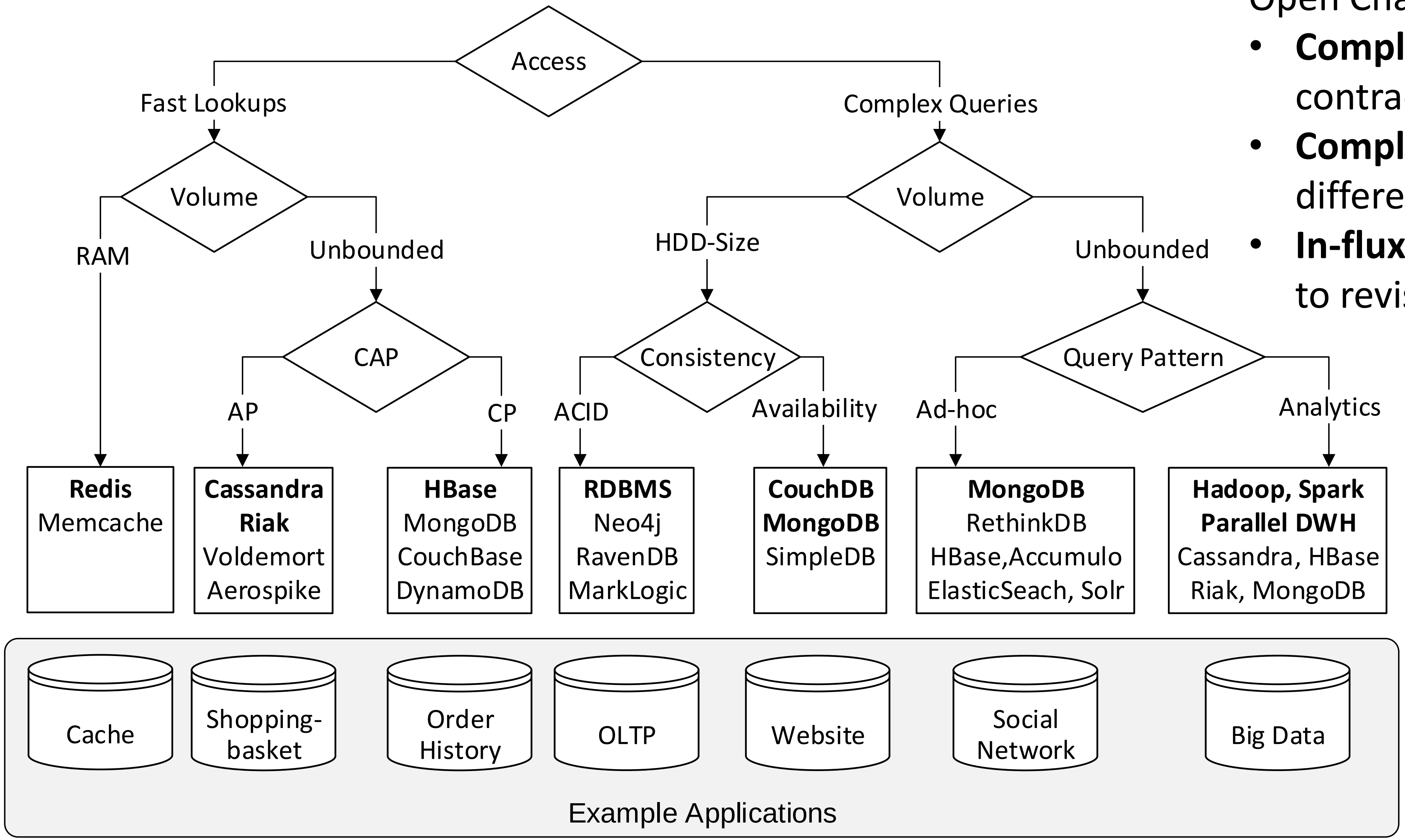


NoSQL Toolbox: Requirements vs. Techniques



https://www.springer.com/gp/book/9783030435059

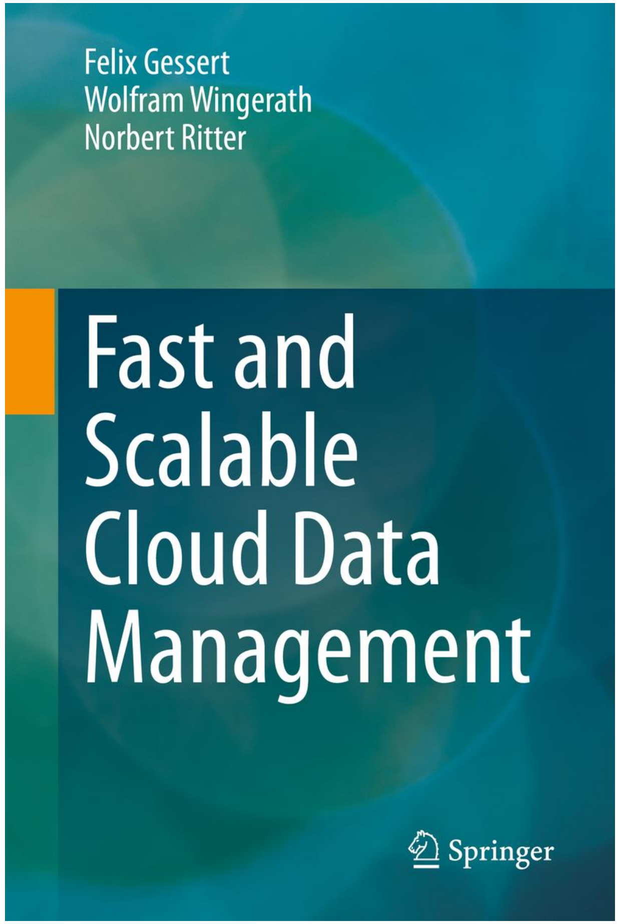
NoSQL Decision Tree

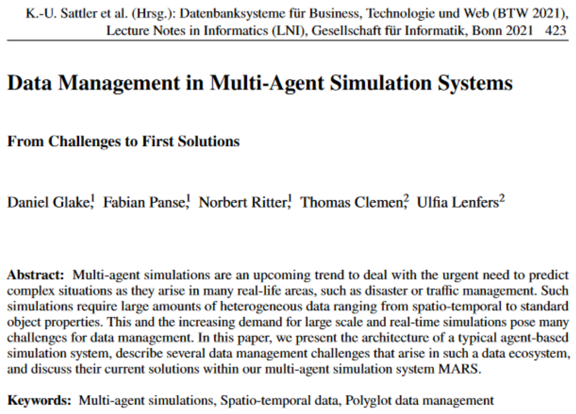


Open Challenges:

- **Complex Trade-Offs** that may contradict one another
- **Complex Architectures** with many different data management systems
- **In-flux Requirements**: You may have to revisit your decision over time

More on the Topic




K.-U. Sattler et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2021), Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 423

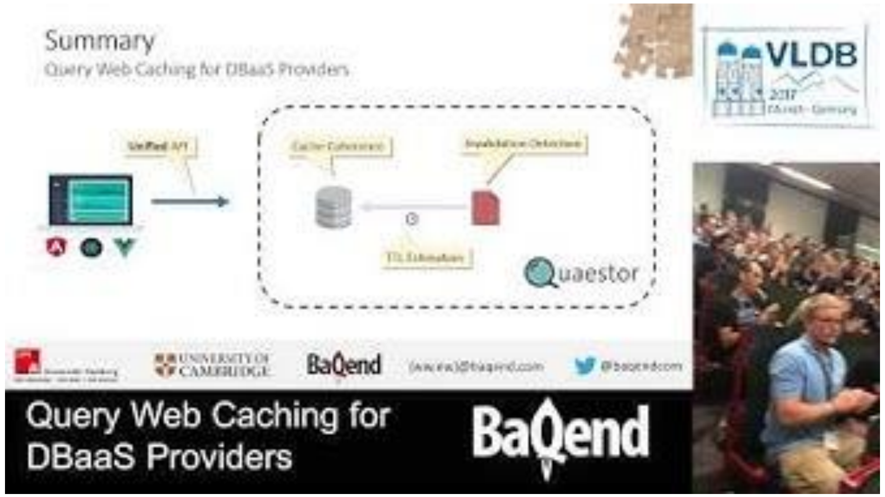
Data Management in Multi-Agent Simulation Systems

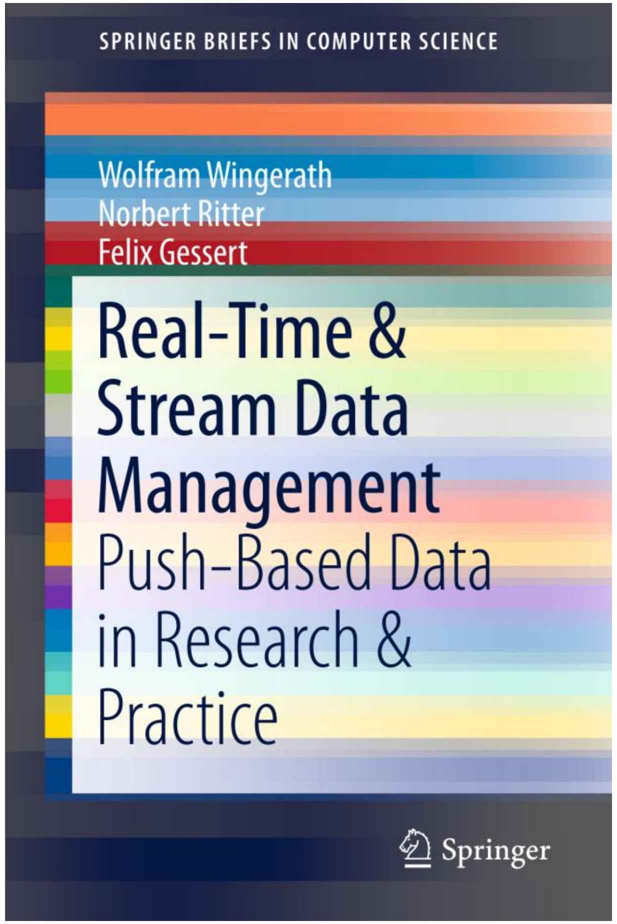
From Challenges to First Solutions

Daniel Glake[1] Fabian Panse[1] Norbert Ritter[1] Thomas Clemen[2] Ulfia Lenfers[2]

Abstract: Multi-agent simulations are an upcoming trend to deal with the urgent need to predict complex situations as they arise in many real-life areas, such as disaster or traffic management. Such simulations require large amounts of heterogeneous data ranging from spatio-temporal to standard object properties. This and the increasing demand for large scale and real-time simulations pose many challenges for data management. In this paper, we present the architecture of a typical agent-based simulation system, describe several data management challenges that arise in such a data ecosystem, and discuss their current solutions within our multi-agent simulation system MARS.

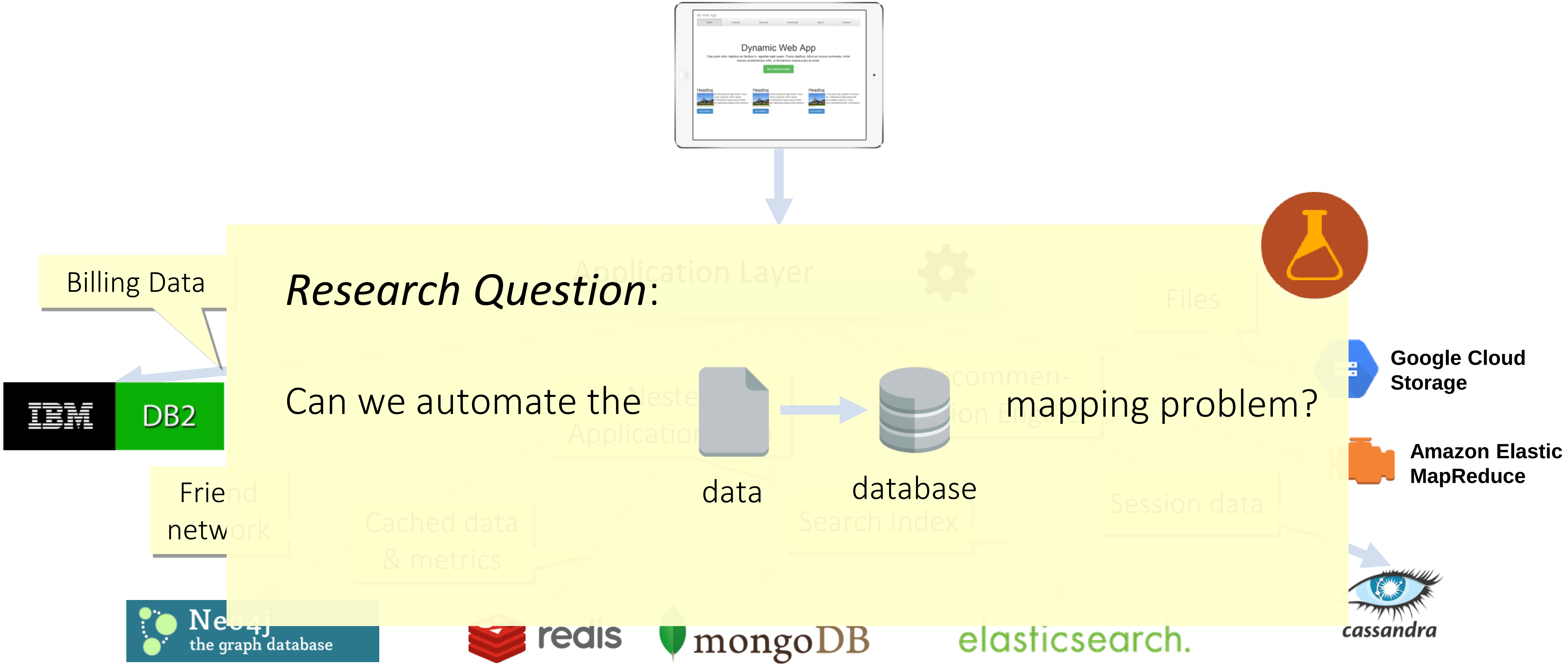
Keywords: Multi-agent simulations, Spatio-temporal data, Polyglot data management





For videos & books, visit dbis.hamburg!

Actual Question: How to Build a System That Does All This?
~~How to Choose The „Right“ Database System?~~



Part II

Terminology & Taxonomies

Terminology & Taxonomies

Polyglot Persistence

Federating (Specialized) Data Stores

Multistores

Modern Federated Database Systems

Hybrid Stores

Polystores

Multidatabases

?????

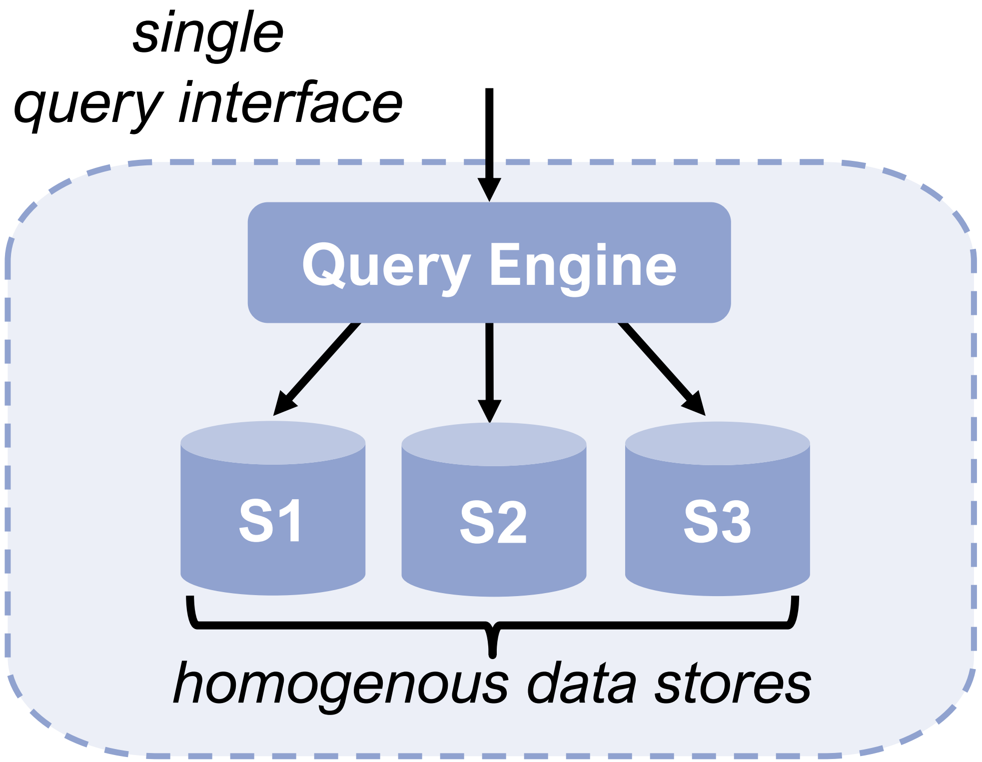
Generalized Data Federation

Query & Data Store based Taxonomy

- **Query Interfaces:** Single | Multiple
- **Data Stores:** Homogenous | Heterogenous

Query & Data Store based Taxonomy

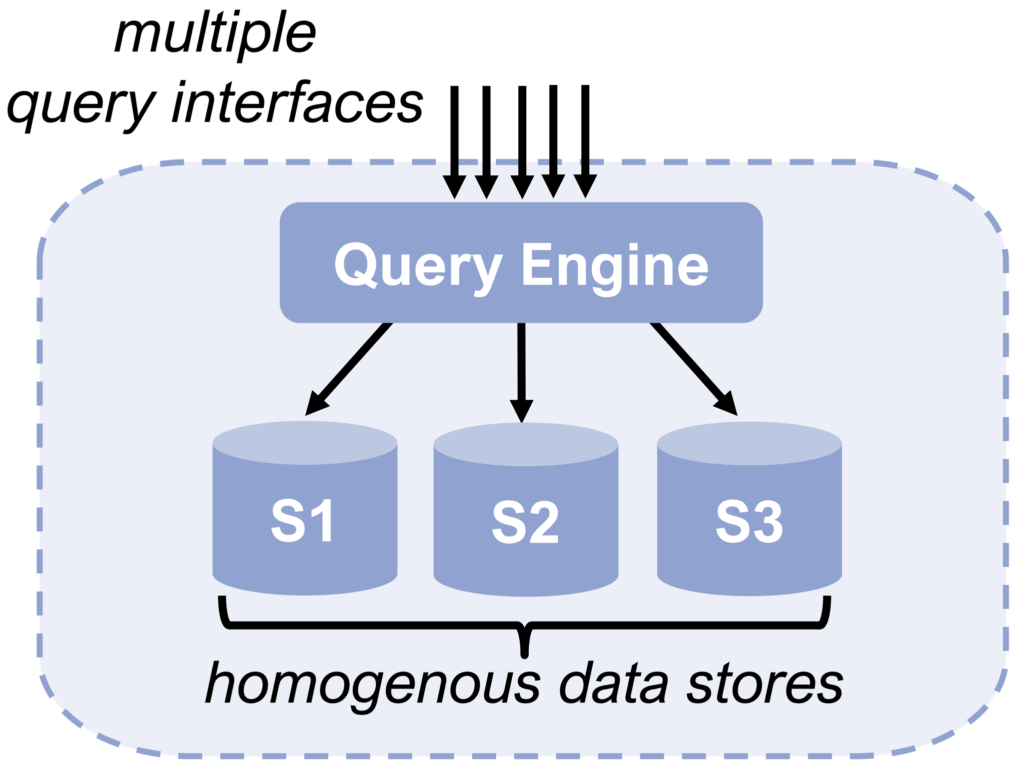
- **Query Interfaces:** Single | Multiple
- **Data Stores:** Homogenous | Heterogenous



- **Federated DB System**
  Single interface, homogenous stores

Query & Data Store based Taxonomy

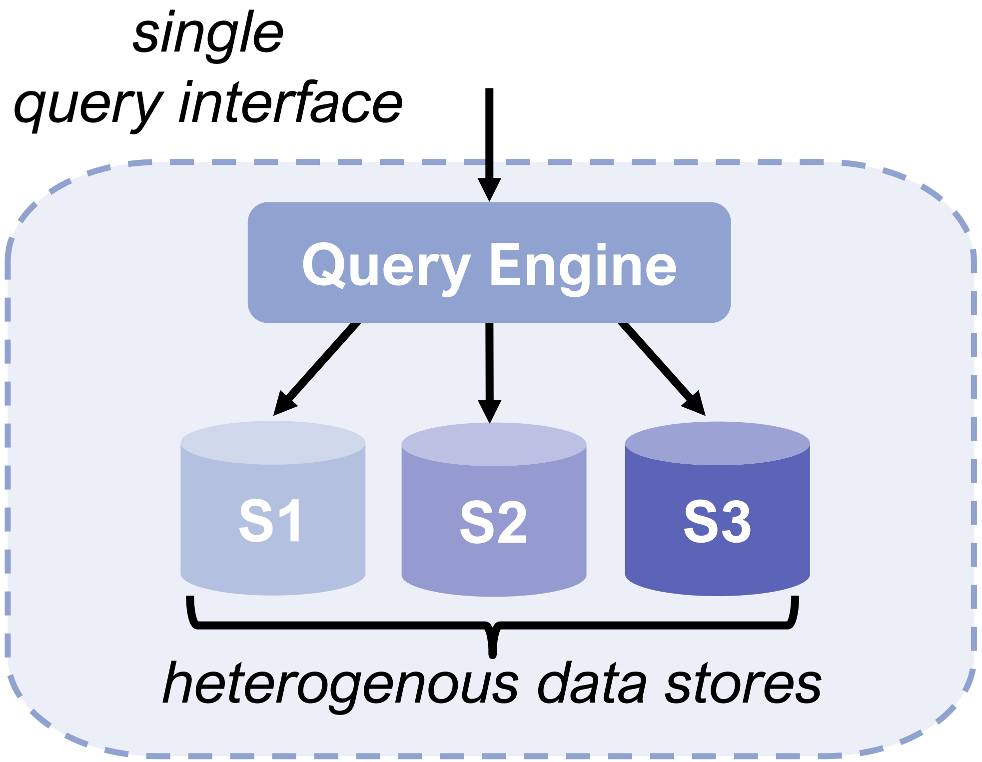
- **Query Interfaces:** Single | **Multiple**
- **Data Stores:** **Homogenous** | Heterogenous



- **Federated DB System**
  Single interface, homogenous stores
- **Polylingual / Polyglot DB System**
  Multiple interfaces, homogenous stores

Query & Data Store based Taxonomy

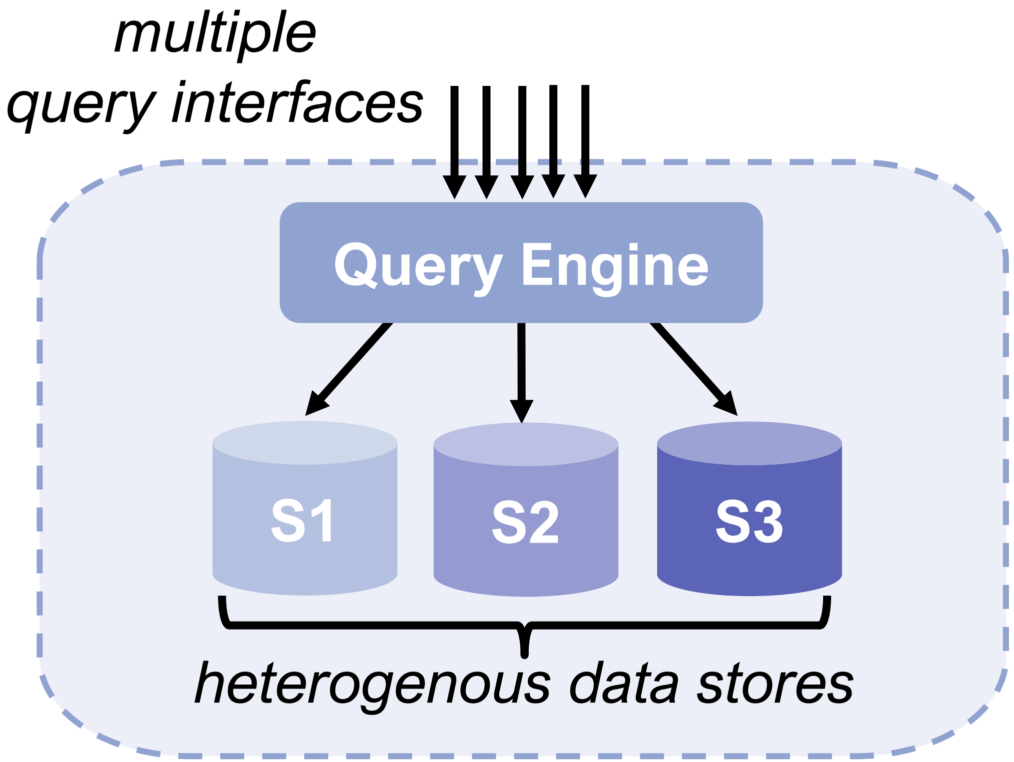
- **Query Interfaces:** Single | Multiple
- **Data Stores:** Homogenous | Heterogenous



- **Federated DB System**
  Single interface, homogenous stores
- **Polylingual / Polyglot DB System**
  Multiple interfaces, homogenous stores
- **MultiStore**
  Single interface, heterogenous stores

Query & Data Store based Taxonomy

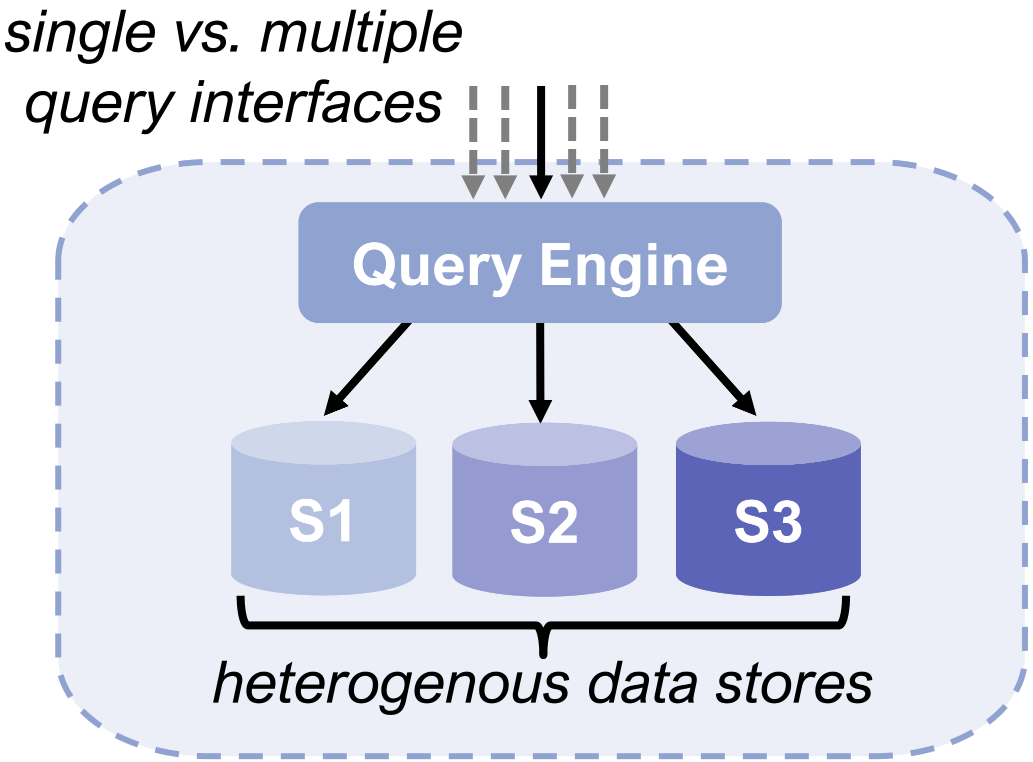
- **Query Interfaces:** Single | **Multiple**
- **Data Stores:** Homogenous | **Heterogenous**



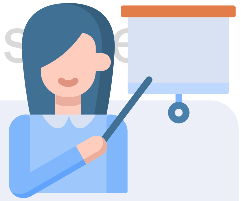
- **Federated DB System**
  Single interface, homogenous stores
- **Polylingual / Polyglot DB System**
  Multiple interfaces, homogenous stores
- **MultiStore**
  Single interface, heterogenous stores
- **PolyStore**
  Multiple interfaces, heterogenous stores

Query & Data Store based Taxonomy

- **Query Interfaces:** Single | Multiple
- **Data Stores:** Homogenous | Heterogenous

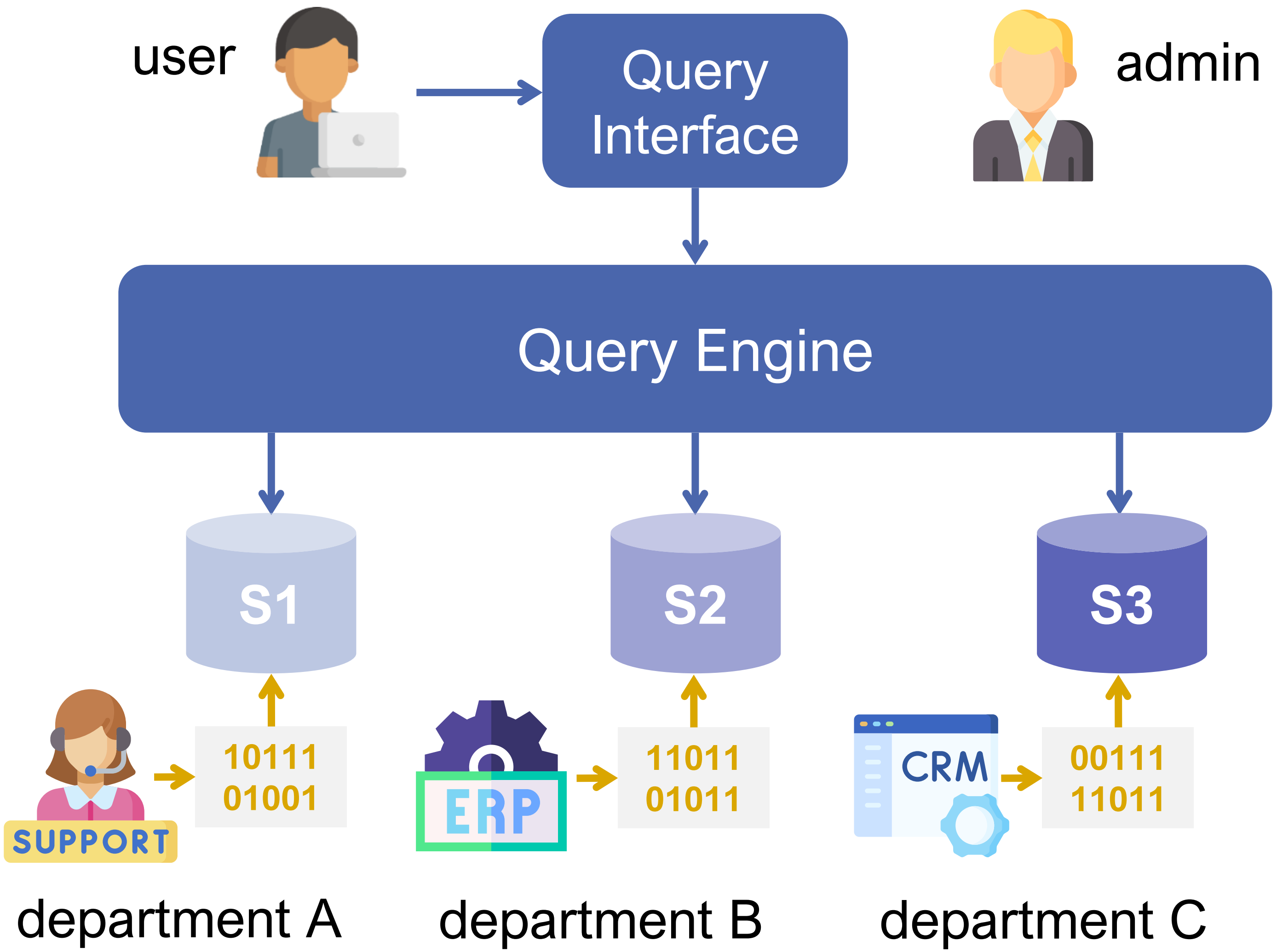


- **Federated DB System**
  Single interface, homogenous stores
- **Polylingual / Polyglot DB System**
  Multiple interfaces, homogenous s



- **MultiStore**
  Single interface, heterogenous stores
- **PolyStore** POLYGLOT
  Multiple interfaces, heterogenous stores

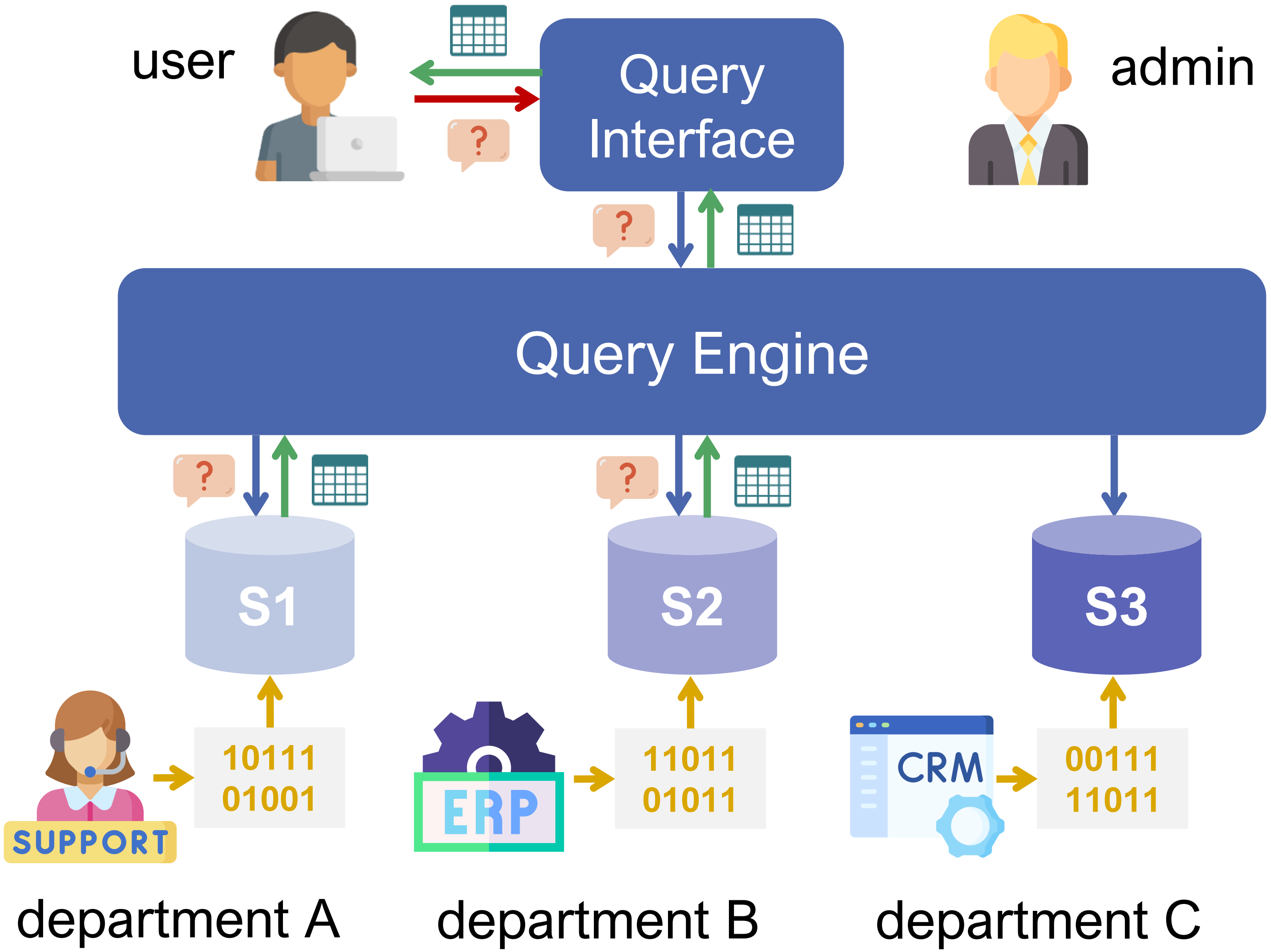
Loosely vs. Tightly Coupled



Loosely
(like a data integration scenario)

Data:	External
Store Autonomy:	High
Access:	Read-Only
Local Config.:	No
Data Quality:	Little control
Semantic Het.:	Possible

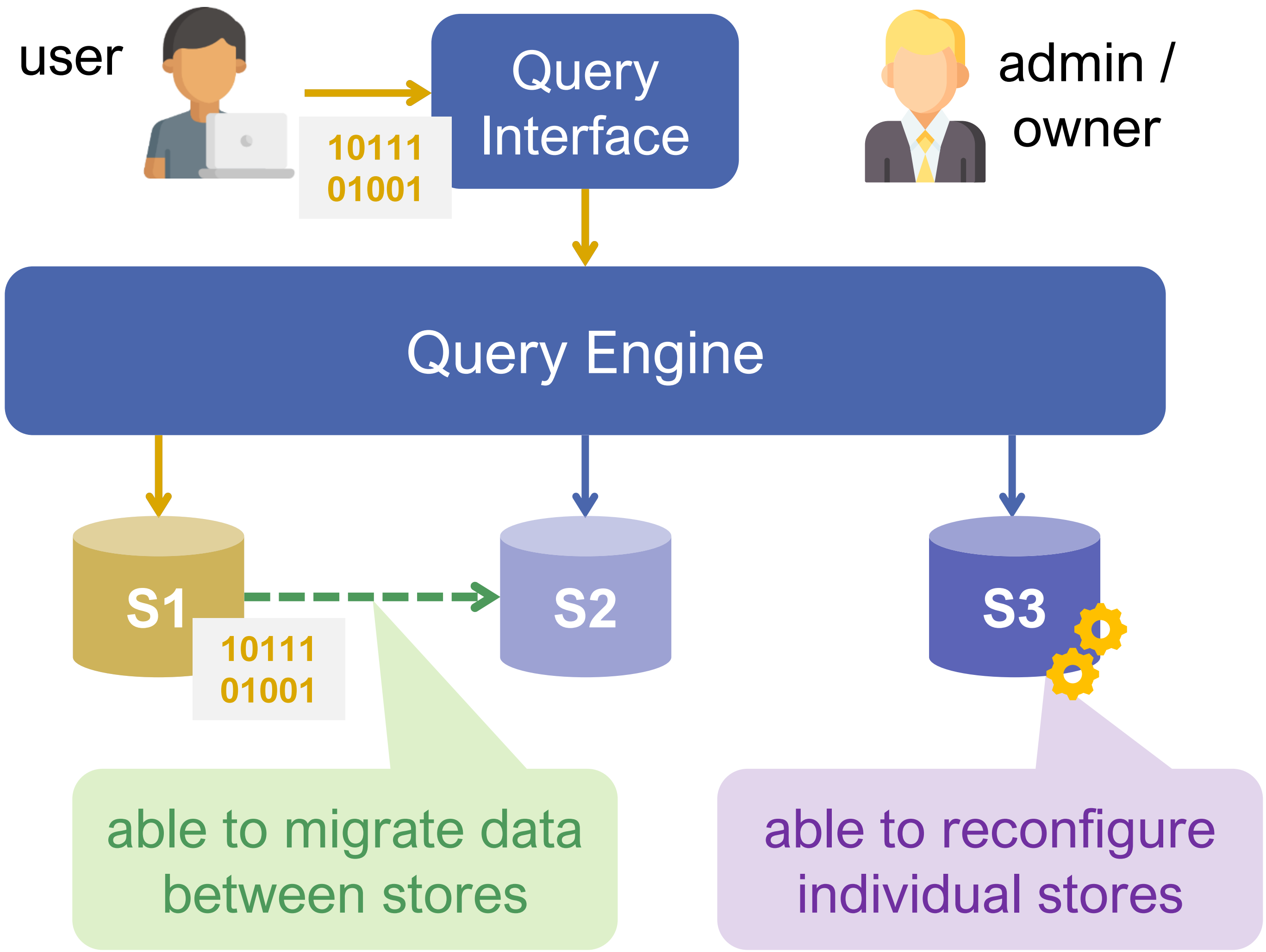
Loosely vs. Tightly Coupled



Loosely
(like a data integration scenario)

Data:	External
Store Autonomy:	High
Access:	Read-Only
Local Config.:	No
Data Quality:	Little control
Semantic Het.:	Possible

Loosely vs. Tightly Coupled



Tightly
(like a distributed system)

Data:	Internal
Store Autonomy:	None
Access:	Write & Read
Local Config.:	Yes
Data Quality:	High control
Semantic Het.:	None

Evaluation Framework

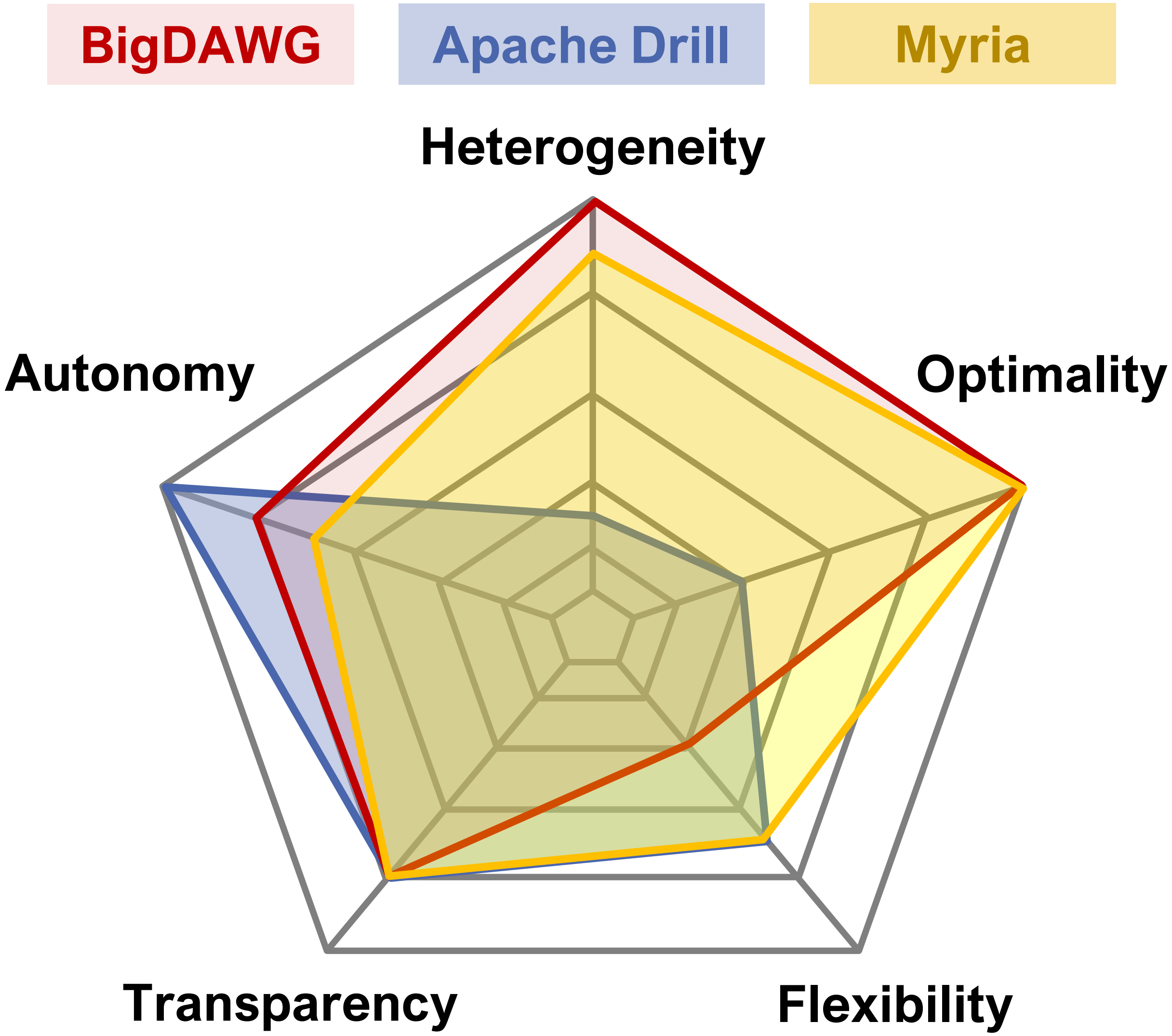
Heterogeneity
Data Stores, Processing Engines & Query Interfaces

Autonomy
Association, Execution & Evolution

Transparency
Location & Transformation

Flexibility
Schema, Interface & Architectural

Optimality
Federated Plan & Data Placement

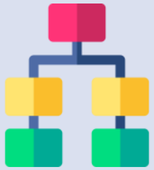


Tan et al., Enabling Query Processing across Heterogeneous Data Models: A Survey, IEEE BigData, 2017.

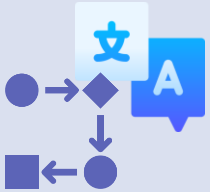
Part III

Basic Techniques & Concepts

Basic Techniques & Concepts: Overview



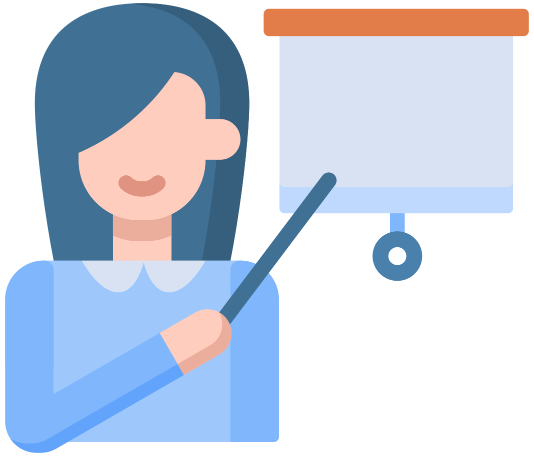
Mediator-Wrapper Architecture
Popular architecture of integration systems



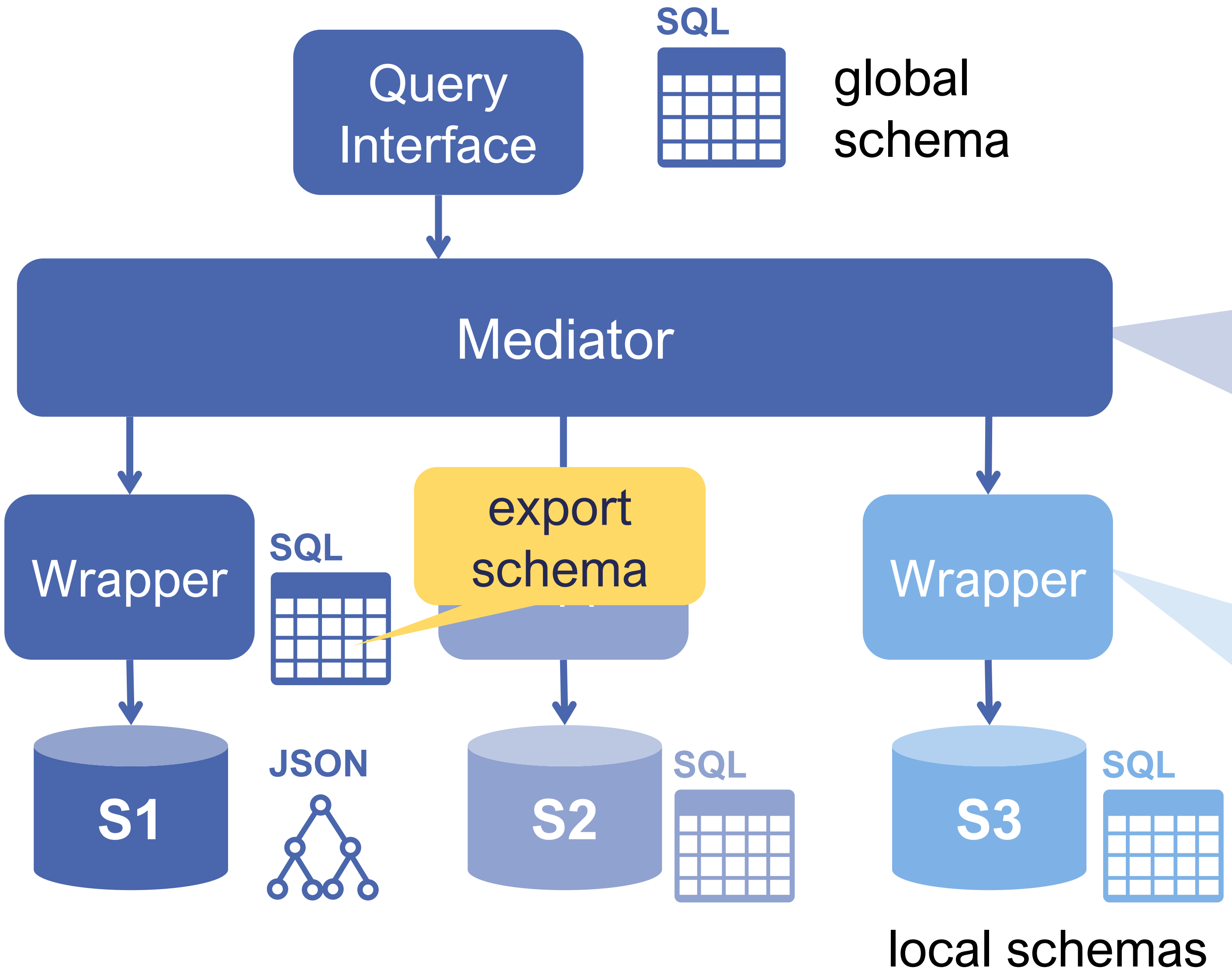
Schema Mapping Languages
How to model relationships between schemas?

Joins
How to combine data from different stores?

Cross-Platform Query Planning
How to optimize queries across stores?



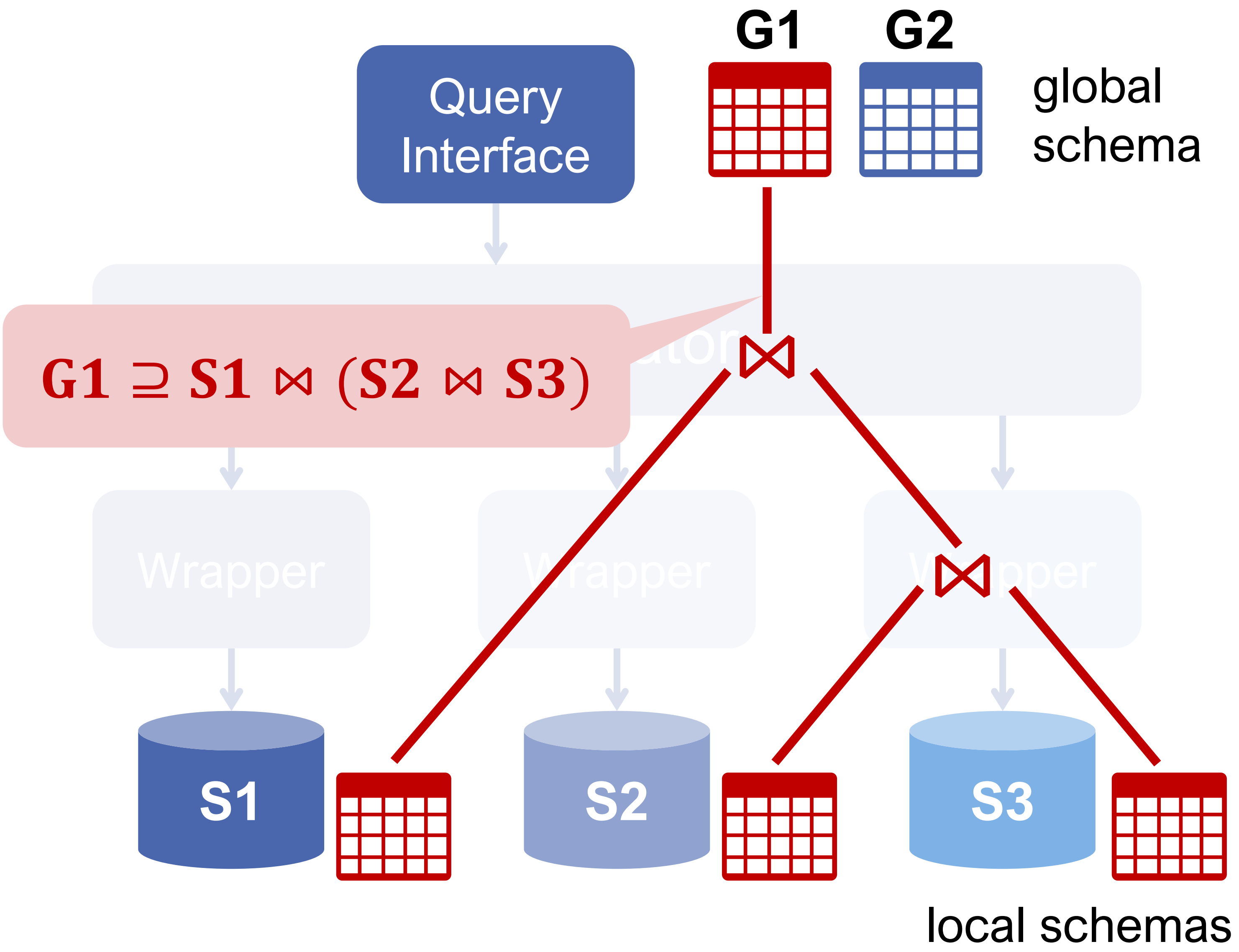
Mediator-Wrapper Architecture



Tasks:
- build query plan
  - decompose query
  - determine join order
- push subqueries
- orchastrate query execution
- combine subquery results

Tasks:
- translate query in local language / schema
- transform result records to global schema

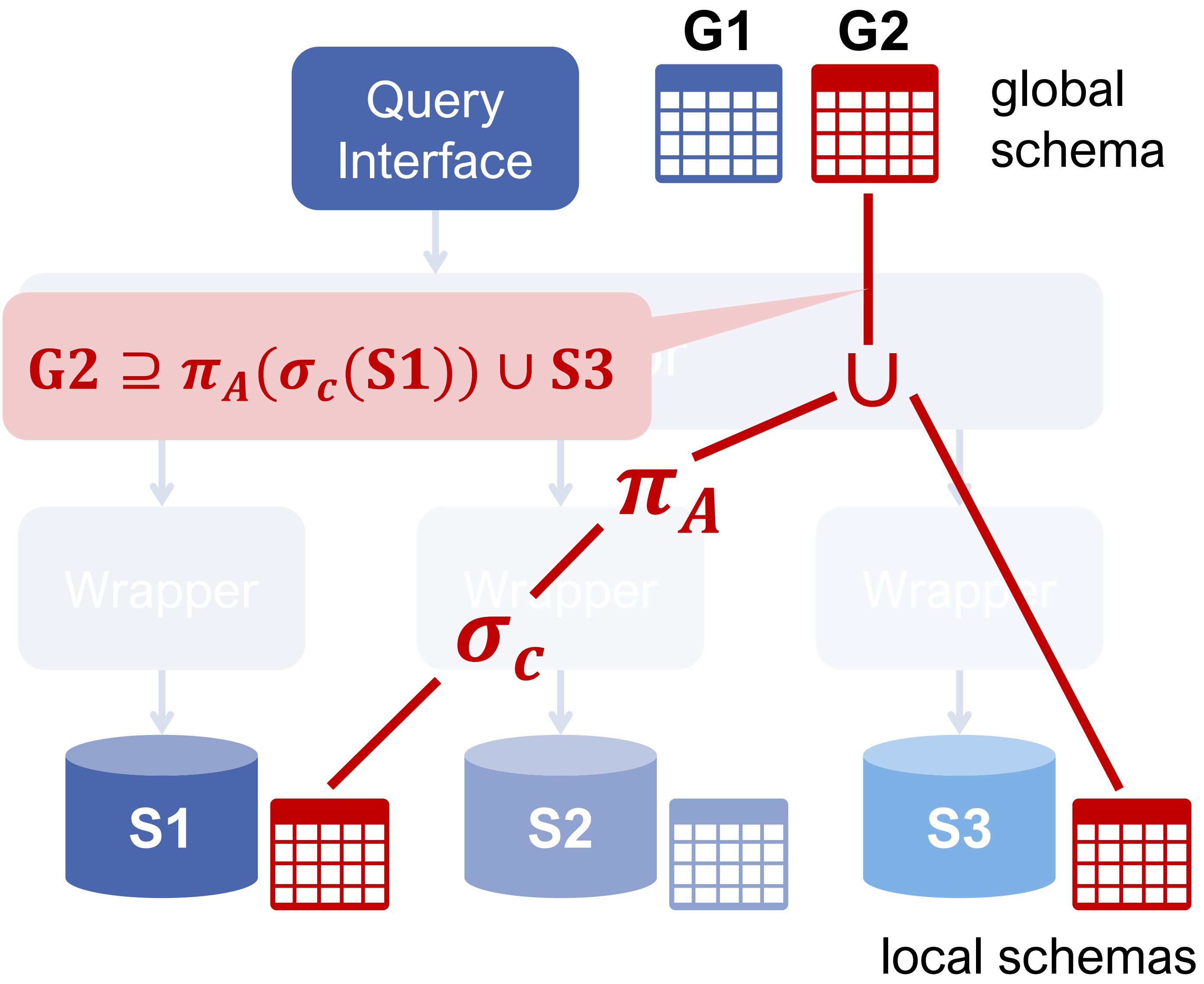
Schema Mapping Languages



Global-as-View (GaV)

Flexibility:	Low
Query Processing:	Simple
Number Views:	Low
Complexity Views:	High
Modeling Power:	Medium

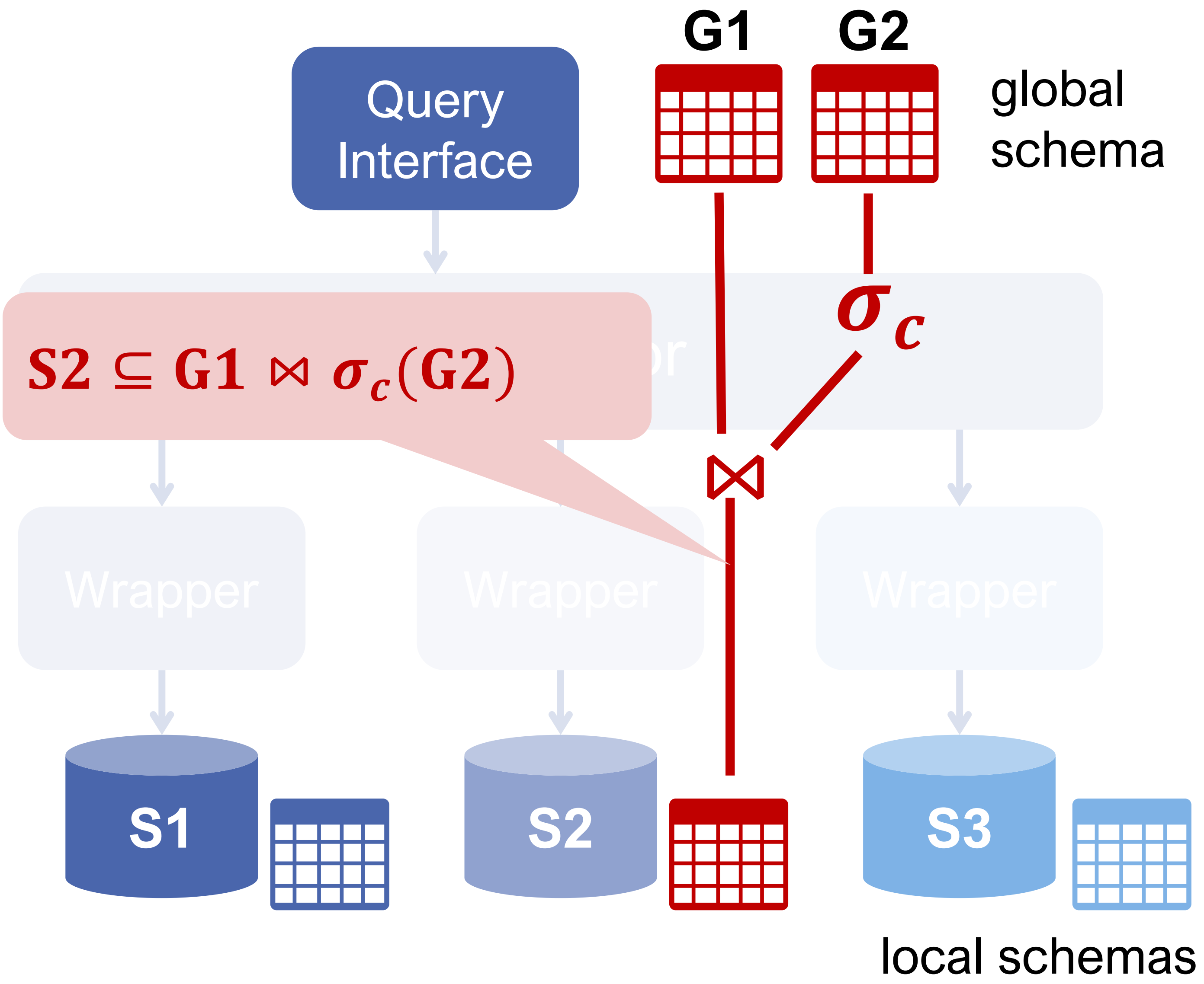
Schema Mapping Languages



Global-as-View (GaV)

Flexibility:	Low
Query Processing:	Simple
Number Views:	Low
Complexity Views:	High
Modeling Power:	Medium

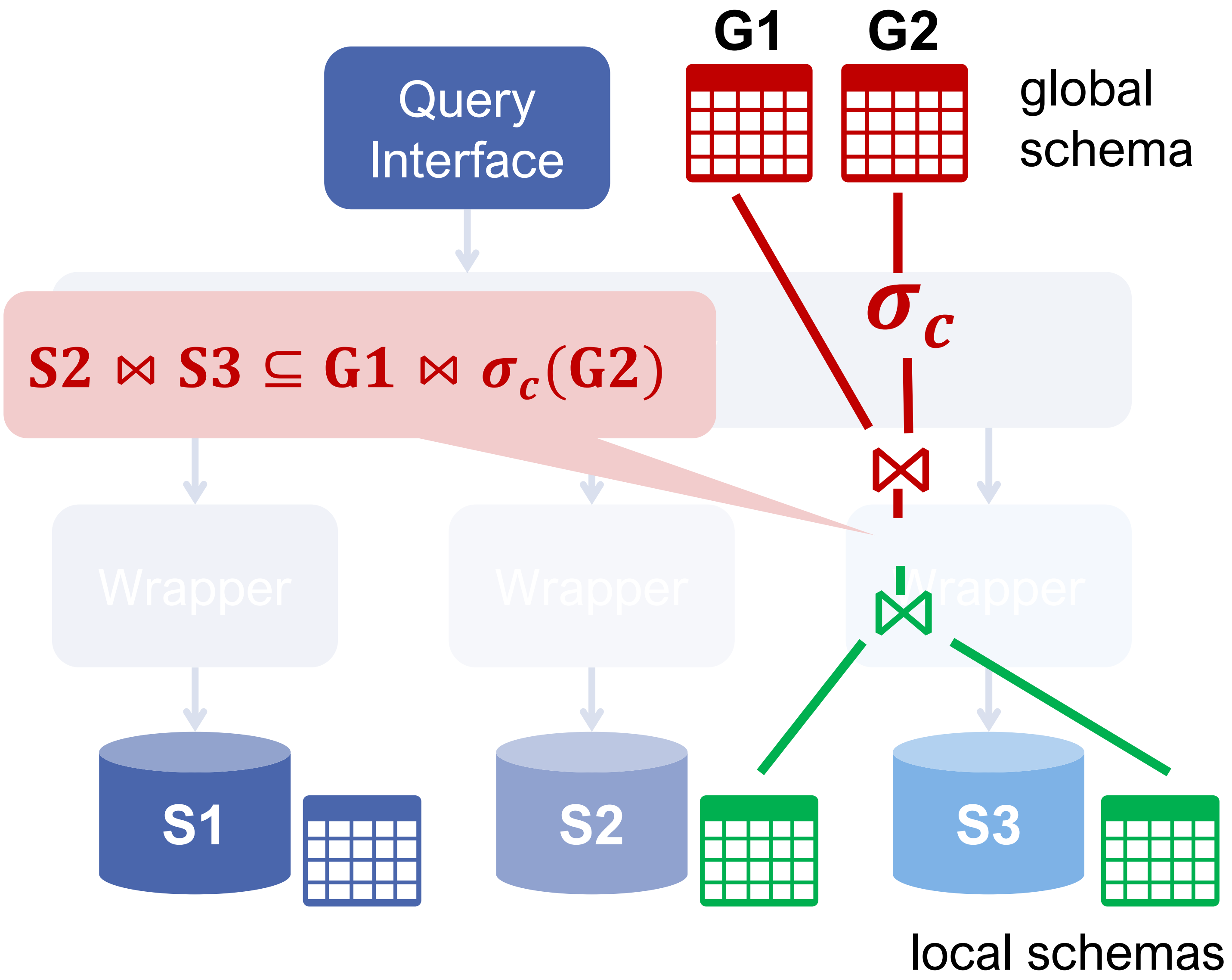
Schema Mapping Languages



Local-as-View (LaV)

Flexibility:	High
Query Processing:	Complex
Number Views:	High
Complexity Views:	Low
Modeling Power:	Medium

Schema Mapping Languages



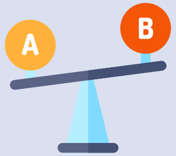
Global-Local-as-View (GLaV)

Flexibility:	High
Query Processing:	Complex
Number Views:	Medium
Complexity Views:	Low - High
Modeling Power:	High

Join Operators, Frameworks & Algorithms

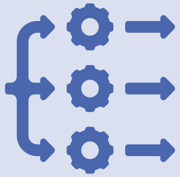
Data Streams

- Ajoin (2021)
- FastJoin (2019)
- ScaleJoin (2016)
- BiStream (2015)
- …



Network & Data Skews

- FastJoin (2019)
- SharesSkew (2018)
- SquirrelJoin (2017)
- Flow-Join (2016)
- …



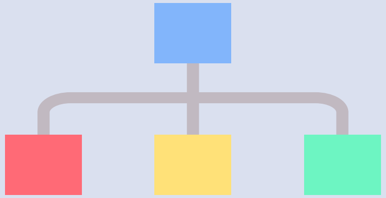
Distributed, Parallel, & GPUs

- GPU-NL Join (2021)
- SquirrelJoin (2017)
- Flow-Join (2016)
- Track Join (2014)
- …

Spatio-temporal

- k-SDJoin (2020)
- HyMJ (2019)
- TL-Join (2019)
- …

Polyglot Data Management

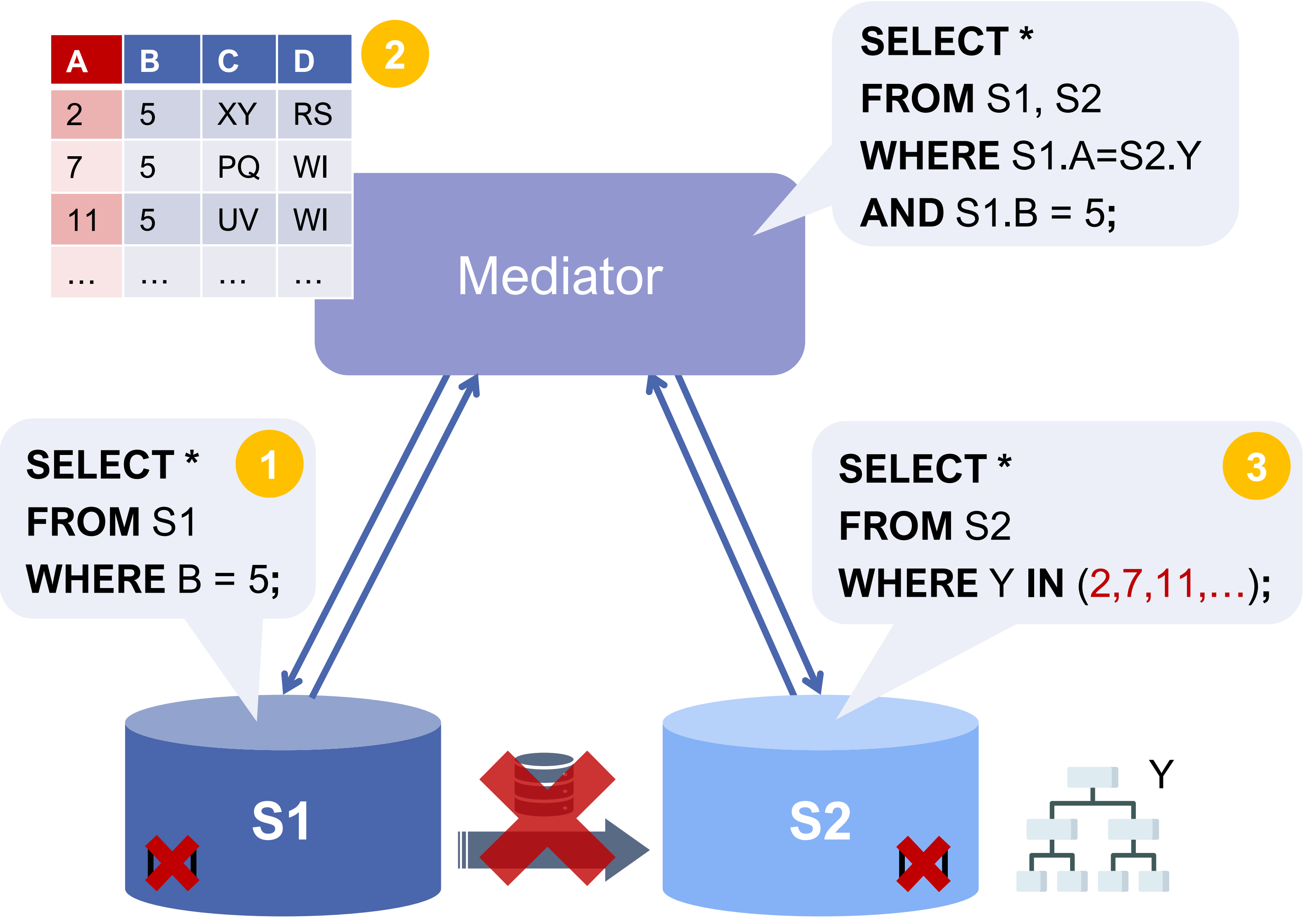


Limited store capabilities:

- Store is not able to perform joins
- Store is not able to provide all records
- Store returns records at irregular frequency

Bind Join (Fetch-Match Strategy)

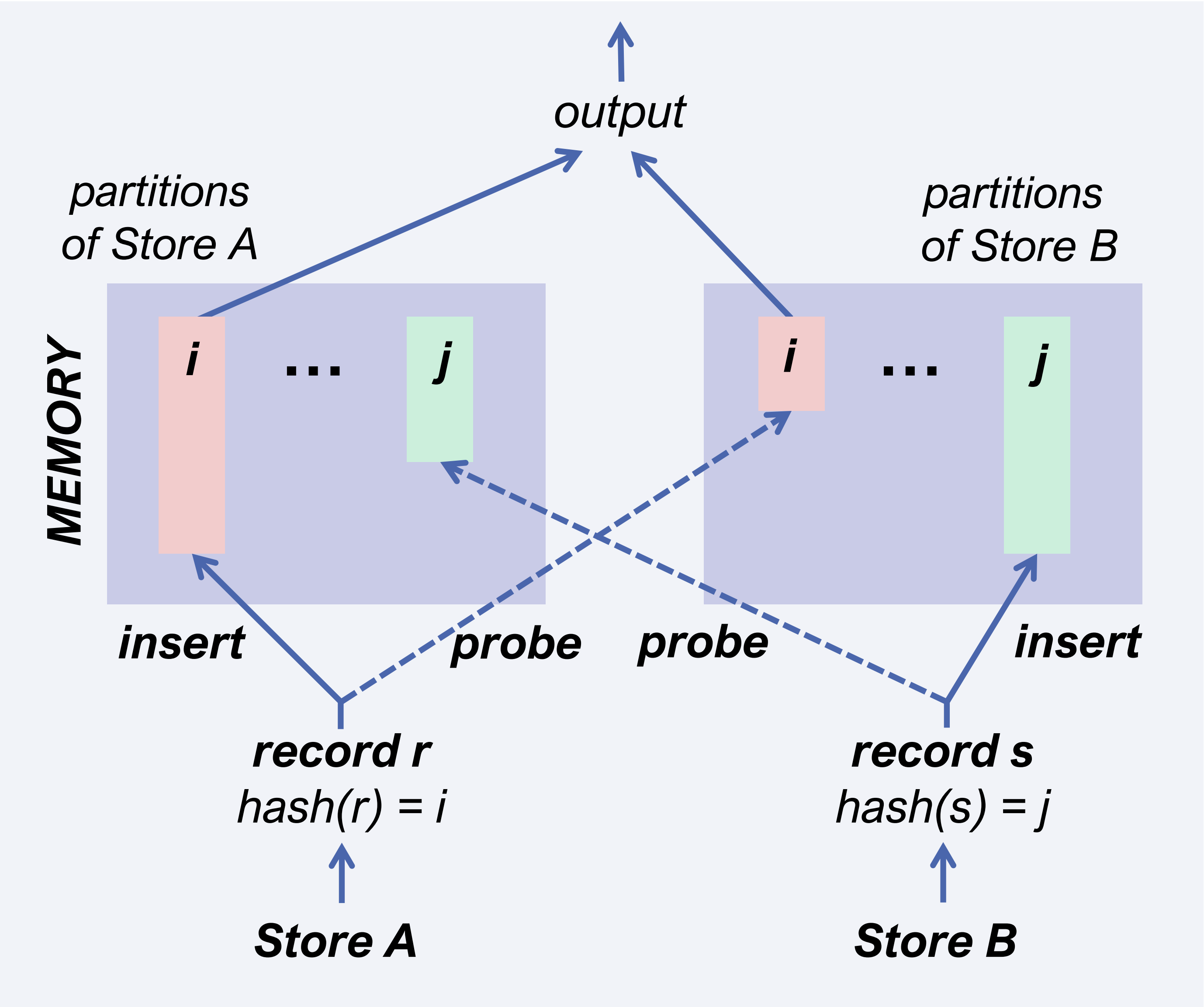
A	B	C	D
2	5	XY	RS
7	5	PQ	WI
11	5	UV	WI
…	…	…	…



- Scenario:
  - Stores are not able to perform joins
  - Mediator cannot ship data between stores
  - |Subquery1| ≪ |Subquery2|
- Without IN-subquery support: One query per join value x ∈ A

  SELECT *
  FROM S2
  WHERE Y = x;
- Fast access per index
- Used in CloudMdsQL and ESTOCADA

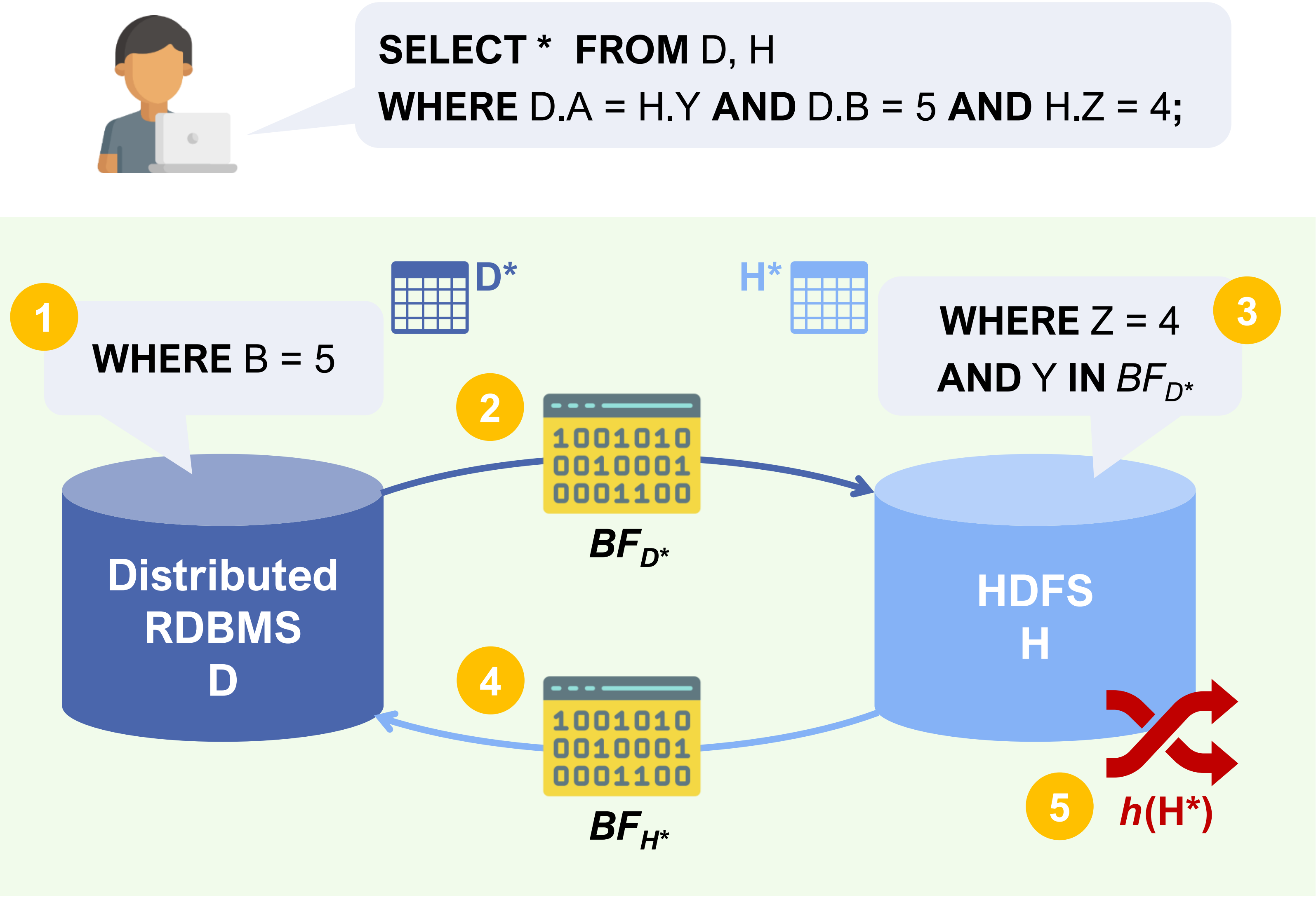
Double Pipelined Symmetric Hash Join



- Using two hash tables
  - insert new record in own hash
  - probe with other hash for potential join partners
- Symmetry reduces risks of blocking
- Pipelining (records are pushed immediately to next operator)
- **XJoin:** Spills partitions to disk
  - Enables larger data sets
  - Allows more parallelism
  - Reduces down times

Urhan et al., XJoin: A Reactively-Scheduled Pipelined Join Operator, IEEE Data Eng. Bull. 23(2), 2000.

ZigZag Join (JEN)



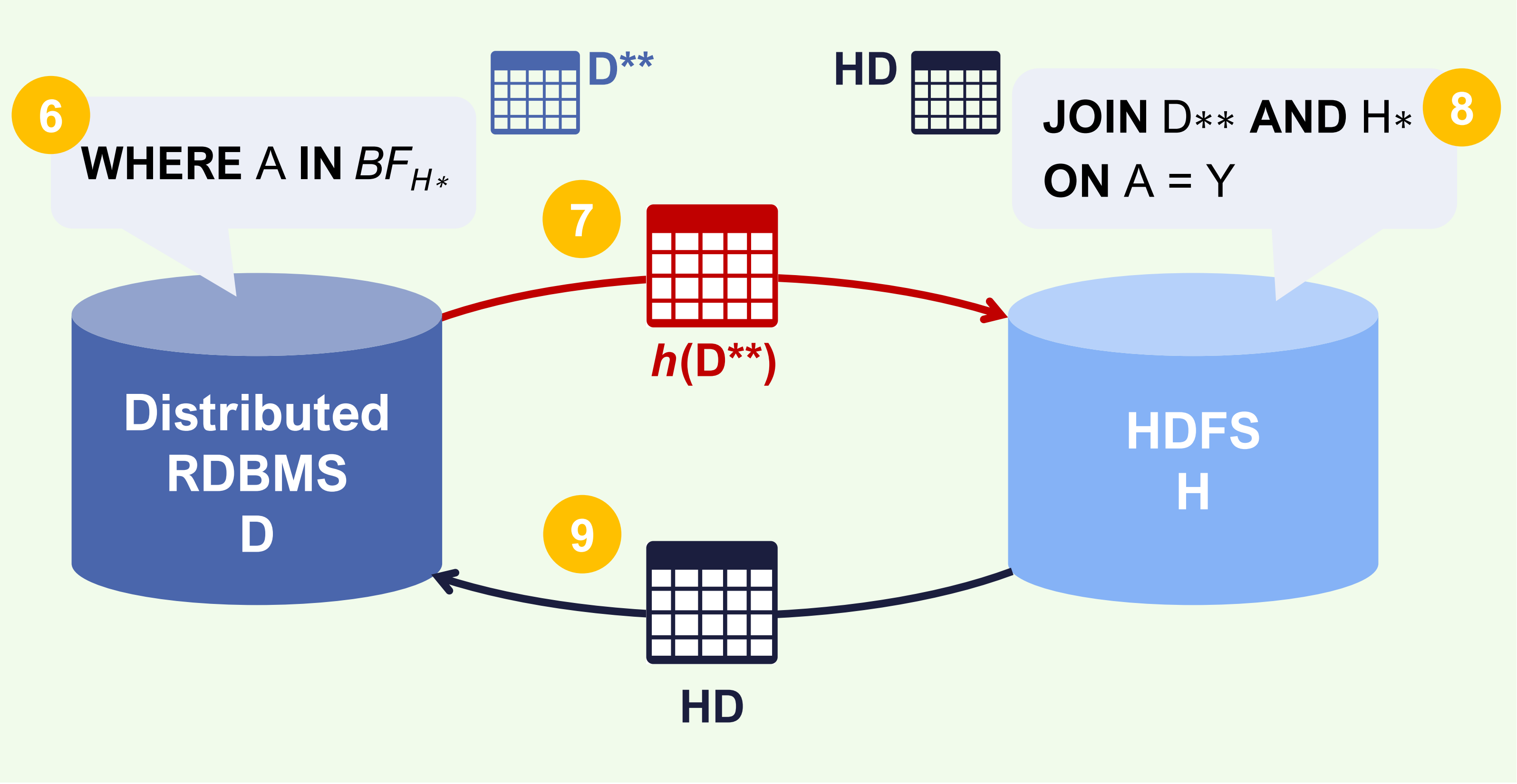
- Join between
  - (distributed) RDBMS D
  - HDFS H
- Assumptions:
  - $|H| \gg |D|$
  - Local predicates not selective
  - Hash *h* for repartitioning

1. Local selection in D
2. Send bloom filter BF_{D*}
3. Local & Join selection in H
4. Send bloom filter BF_{H*}
5. Shuffle H* using *h*

Tian et al., Joins for Hybrid Warehouses: Exploiting Massive Parallelism in Hadoop and Enterprise Data Warehouses, EDBT, 2015.

ZigZag Join (JEN)

SELECT * FROM D, H
WHERE D.A = H.Y **AND** D.B = 5 **AND** H.Z = 4;



- Join between
  - (distributed) RDBMS D
  - HDFS H
- Assumptions:
  - $|H| \gg |D|$
  - Local predicates not selective
  - Hash *h* for repartitioning

6 Join selection in D

7 Distribute D** to HDFS partitions using *h*

8 Join D** and H* (+ group by & aggregate)

9 Send join result

Tian et al., Joins for Hybrid Warehouses: Exploiting Massive Parallelism in Hadoop and Enterprise Data Warehouses, EDBT, 2015.

Cross-Platform Query Planning

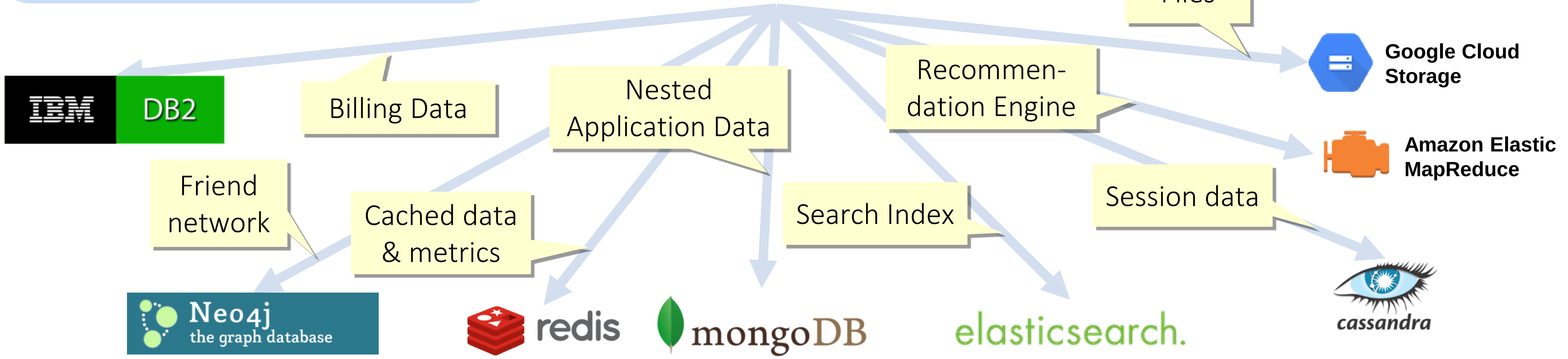
Single Application Query (e.g, Use-Case)

Operator in Query
- Filter
- Map
- Aggregation
- Grouping
- Join
- Path Matches
- …

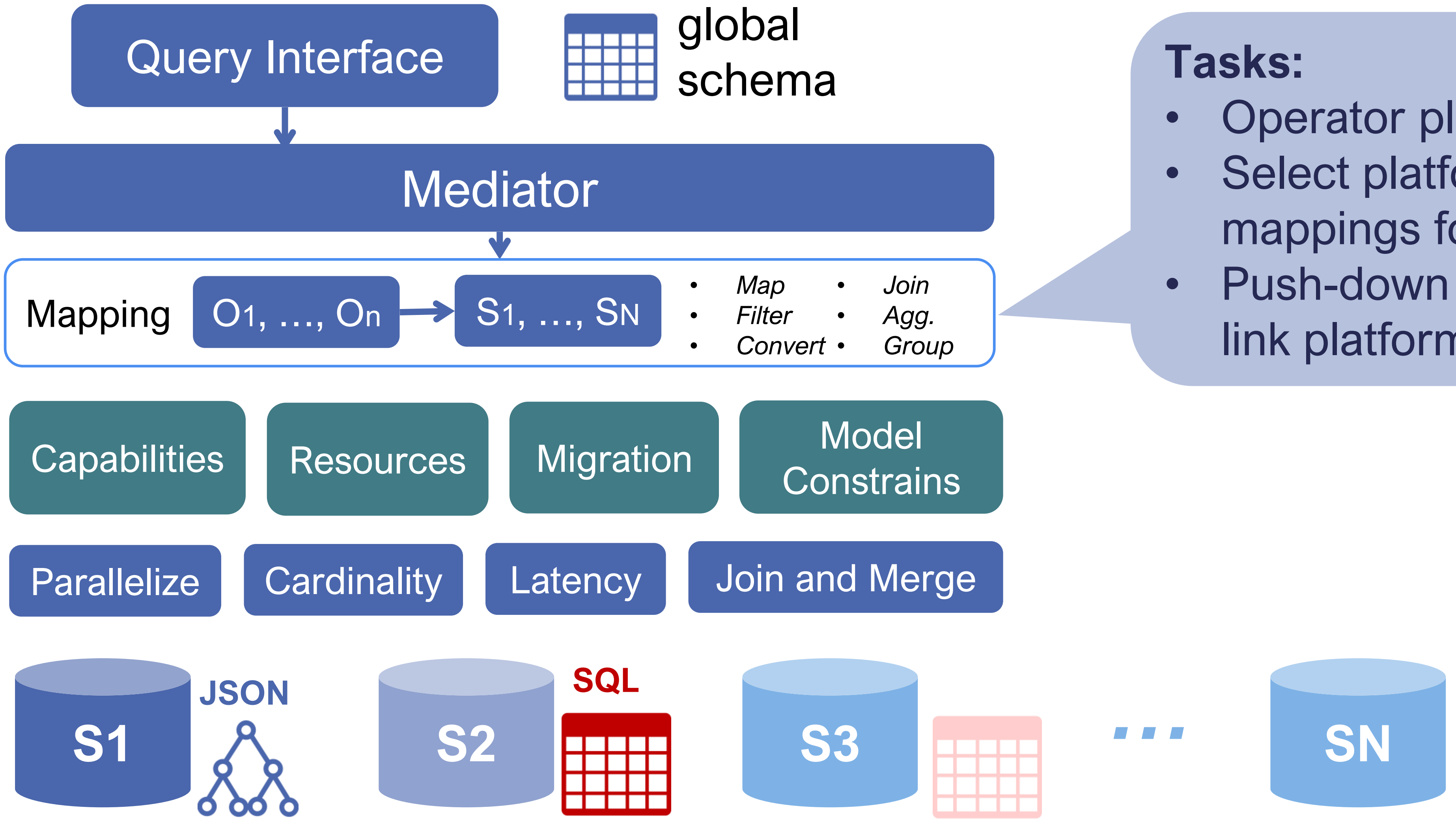
Placement Planning

Planning:
- Map operators to stores
- NP-hard for discrete sets
- Add migrations of data or operators

Constraints:
- Service Level Objectives
- Find efficient Migrations in platform network w.r.t. computing resources



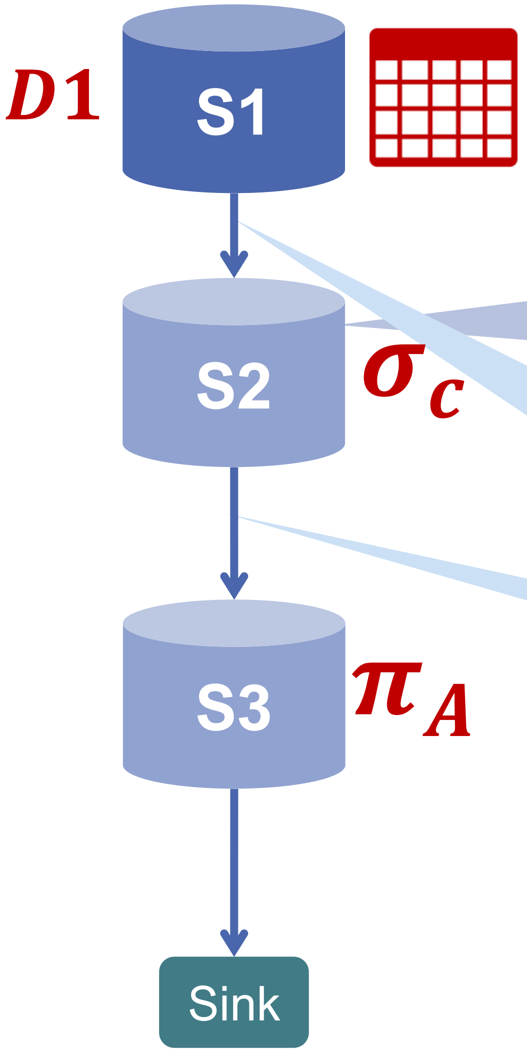
Cross-Platform Query Planning



Tasks:

- Operator plan
- Select platforms with mappings for operations
- Push-down operations and link platforms with migrations

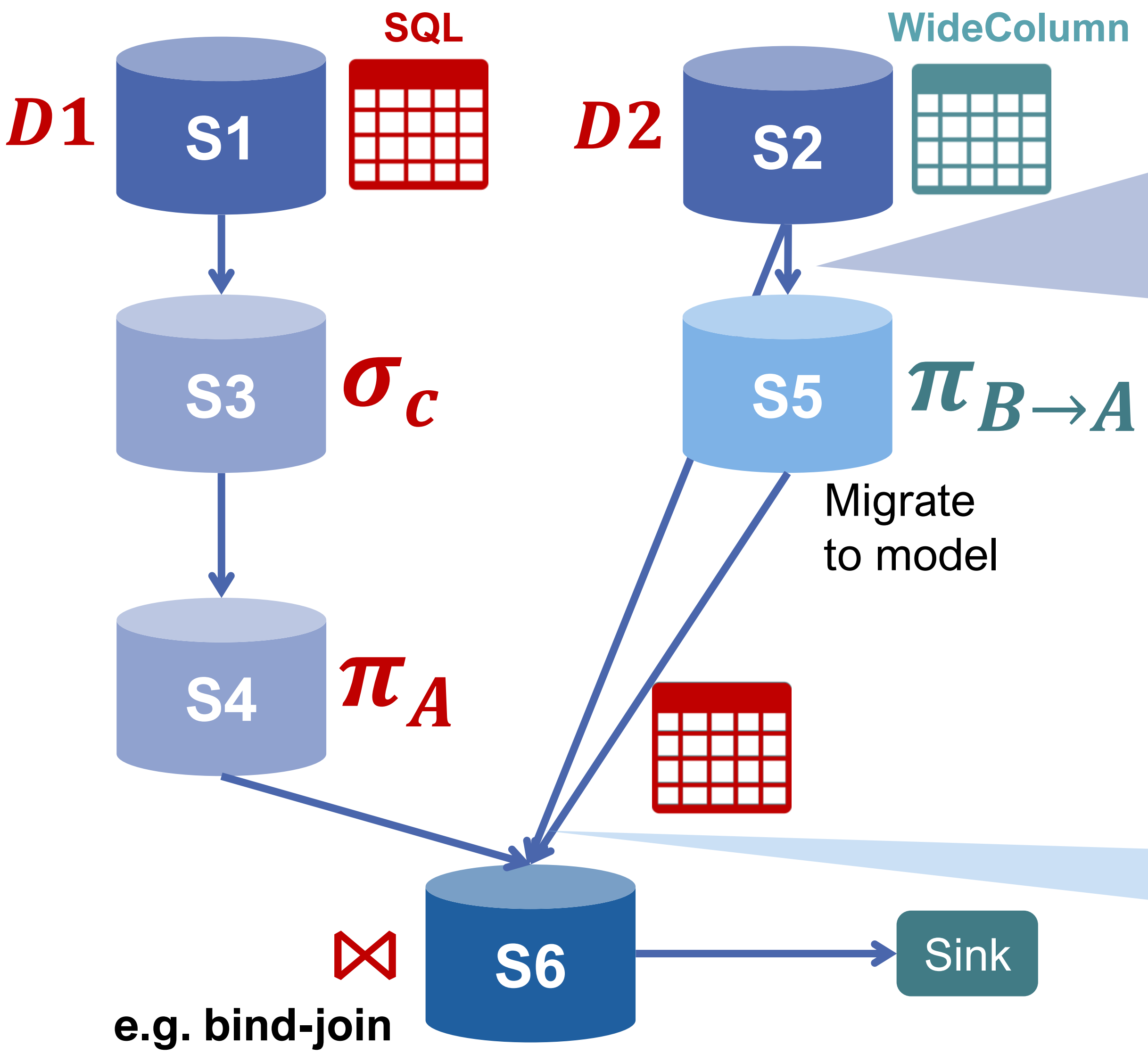
Shapes – Sequential



Hint:
- Layered topology placement
- Single platform support of (special-filter) operations
- Forward result to next store

Network:
- Loading complete data set or
- Use streaming (distributed stream support) – Volcano model

Shapes – Hierarchy



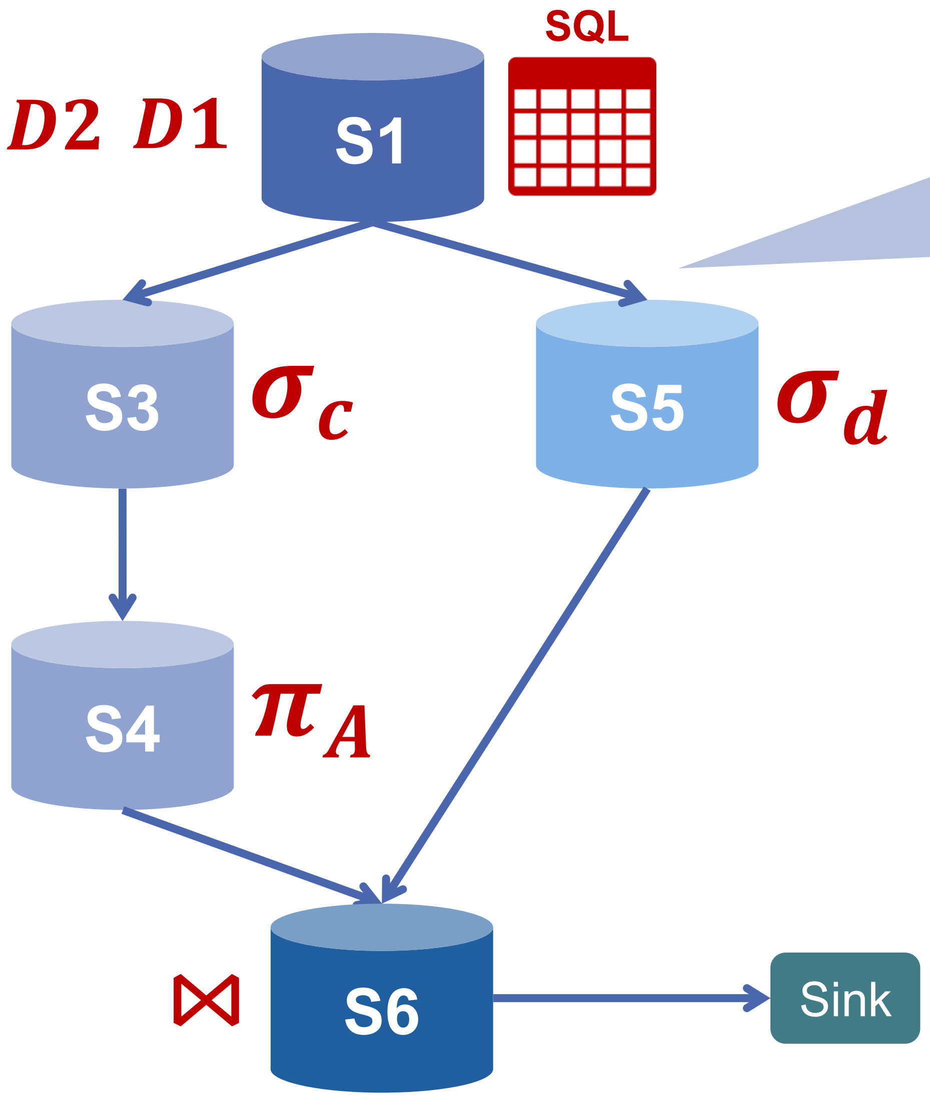
Hint:

- Replicated topology placement
- Decomposition for multiple sources (S1 & S2)
- Parallel execution
- Decomposition under actual system deployment (same node, but different stores)

Network:

- Symmetric vs. asymmetric join placement

Shapes – Diamond

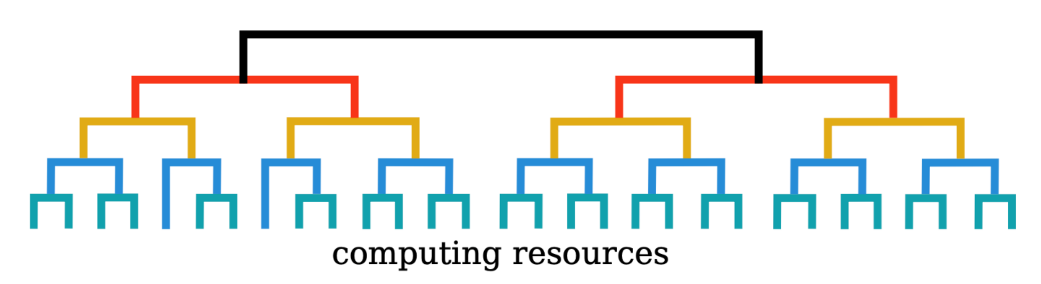


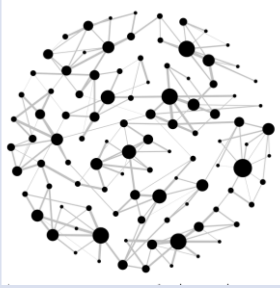
Hint:

- Diamond topology
- Same as hierarchy-shape
- Partitioning of single data sets (e.g., reducing-workload-partitioning)

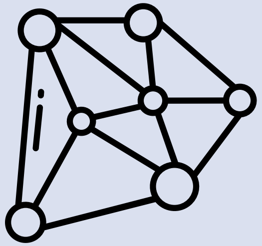
Operator Placement – Approaches

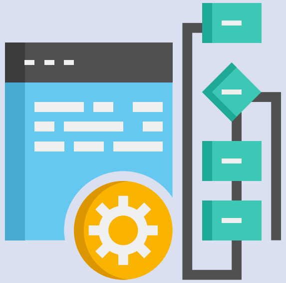
- **Model-based:** different strategies for placement solution
  - Hierarchical Placement
  - Pruned Space Placement
  - Relax-Expand-Solve



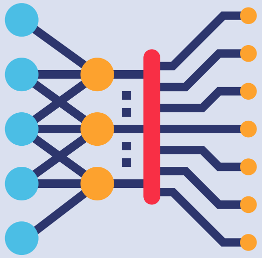


- **Model-free:** provide direct placement seek
  - Greedy First-match
  - Local optimization on greedy-first
  - Tabu search





- **ML-based:**
  - Explore placement decision for similar workloads
  - Learn latency of operator mappings
  - Learn cardinalities of topologies (JOP)



Operator Placement – Model-free Tactics

1. Resolve dependencies between platforms and operators
2. Fixed operators as initial placement and greedy expansion along logical plan
   - Neighbour lookup
3. Co-Locate operators on same platforms
4. Move single operator to another location to reduce estimated cost and latency
   - Local optima problem
   - May split co-location
5. Switch platform by adding migration between source and target
6. Enumerate multiple plans (repeat step 3., 4. and 5. until threshold)
   - Terminate, when no further improvement

Part IV

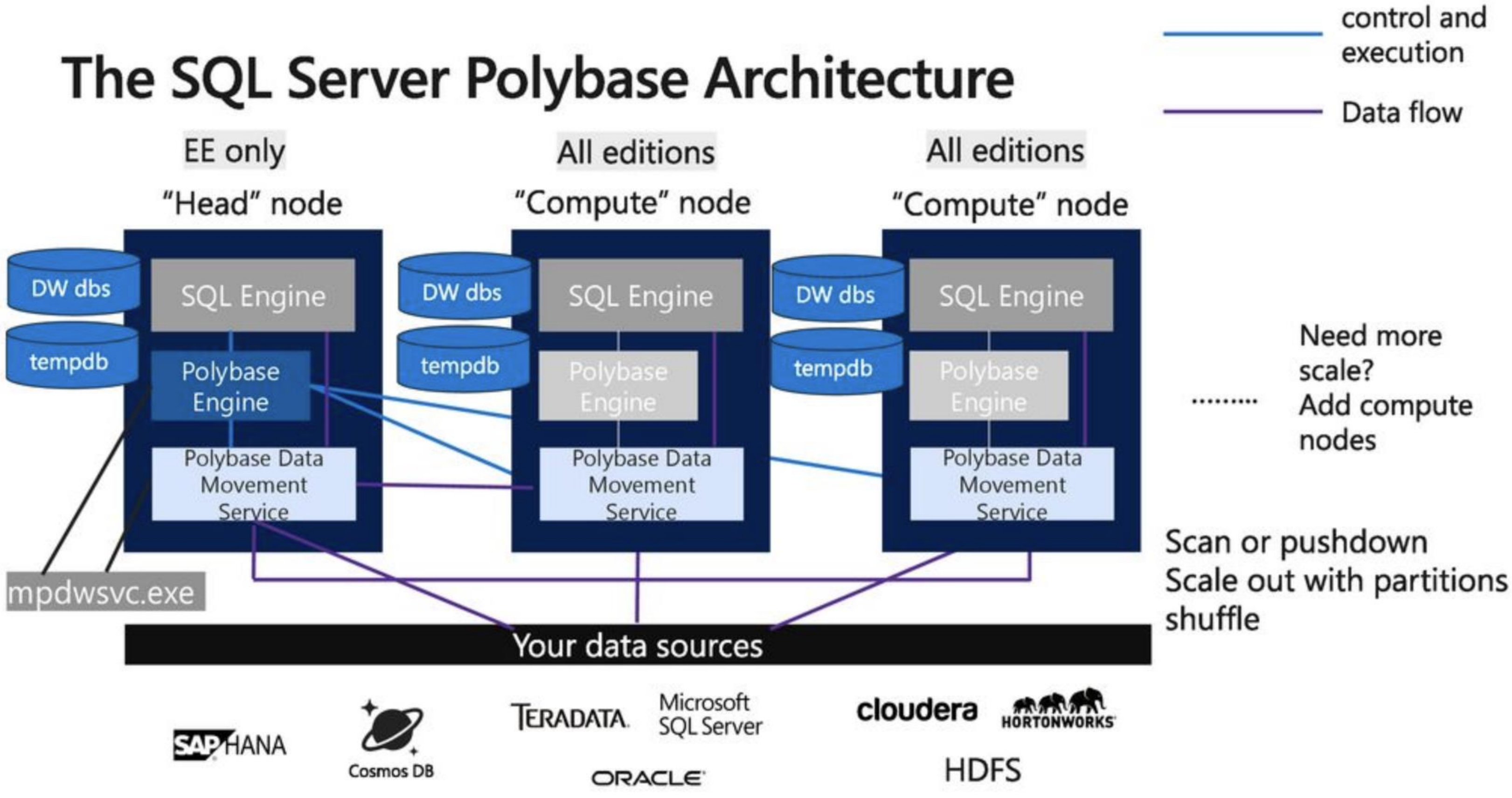
Current Systems

Overview: Polyglot Data Management Systems

	Multistore (Single Interface)	Polystore (Multiple Interfaces)
Loosely coupled	PolyBase BigIntegrator FORWARD Apache Drill (Calcite) QoX QUEPA Odyssey	Myria
Tightly coupled	**RHEEM** MuSQLE HadoopDB	**ESTOCADA** Polypheny-DB
Hybrid	**CloudMdsQL** SparkSQL	**BigDAWG**

PolyBase

- Virtual data integration solution from **Microsoft**
- Distributed compute engine integrated with MS SQL Server
- Query data where it lives (T-SQL):
  - Oracle
  - MongoDB
  - Teradata
  - Hadoop-Cluster
  - Cosmos-DB
  - S3-compatble Store
  - SAP HANA



Example from: PolyBase Extension Group Model: https://docs.microsoft.com/de-de/sql/relational-databases/polybase/polybase-scale-out-groups?view=sql-server-ver16, Accessed: June 2022

PolyBase – Query Concept

- **Manual schema definition by Admin**
- **Create external data source in T-SQL (e.g., MongoDB)**
  - Global schema in MS SQL
  - Definition of relational view on source such as MongoDB collection
  - User-defined statistics for source
  - MS SQL applies flattening rules on hierarchial source models
- **Bridge the heterogeneity of models**

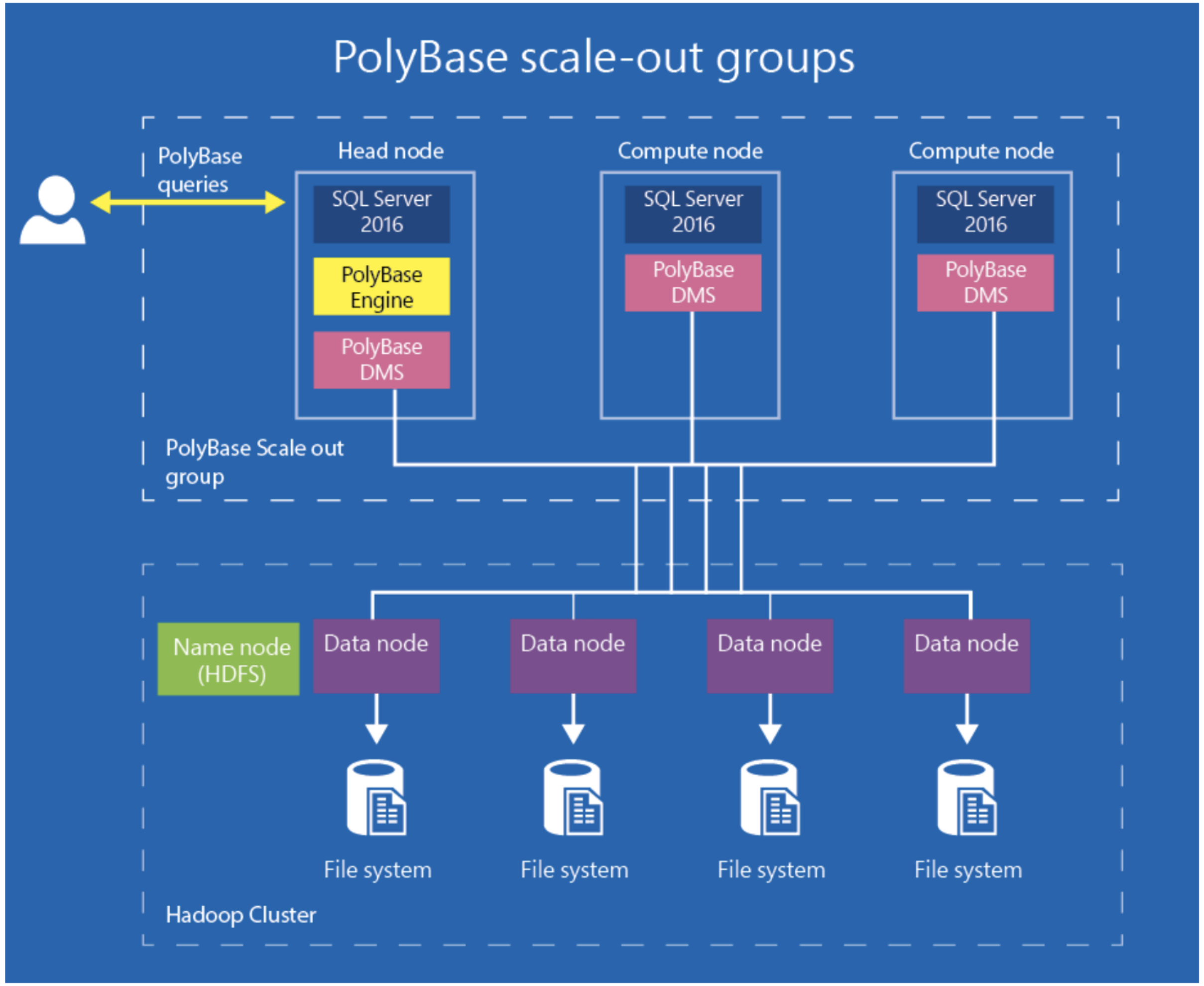
```
CREATE EXTERNAL DATA SOURCE external_data_source_name
WITH (LOCATION = '<mongodb://<server>[:<port>]>'
[ [ , ] CREDENTIAL = <credential_name> ]
[ [ , ] CONNECTION_OPTIONS = '<key_value_pairs>'[,...]]
[ [ , ] PUSHDOWN = { ON | OFF } ])
[ ; ]
```

```
CREATE EXTERNAL TABLE [MongoDbRandomData](
  [_id] NVARCHAR(24) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
  [RandomData_friends_id] INT,
  [RandomData_tags] NVARCHAR(MAX) COLLATE SQL_Latin1_General_CP1_CI_AS)
WITH (
  LOCATION='MyDb.RandomData',
  DATA_SOURCE=[MongoDb])
```

Example from: PolyBase Extension Group Model: https://docs.microsoft.com/de-de/sql/relational-databases/polybase/polybase-scale-out-groups?view=sql-server-ver16, Accessed: June 2022

PolyBase – Optimization Model

- **Distributed query execution across SQL Servers**
  - Read external partitioning metadata
  - Split MS SQL source and remote source
  - Push-down operations where possible
- **Plugin architecture for SQL-Server**
  - Mapping of T-SQL to stores
  - Scale-out compute node
  - PolyBase waits for source data to be processed



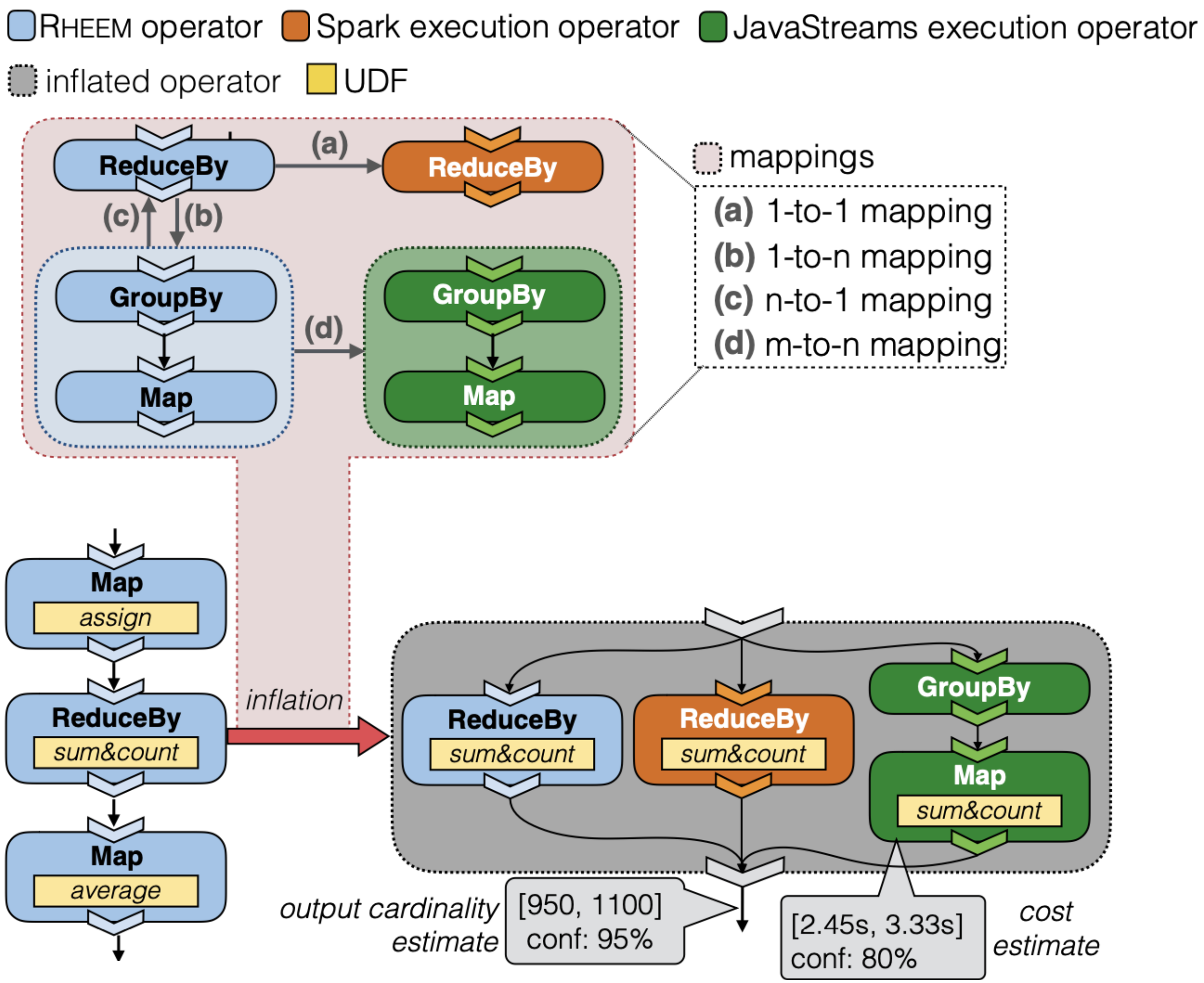
Example from: PolyBase Extension Group Model: https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-scale-out-groups?view=sql-server-ver16, Accessed: August 2022

Overview: Polyglot Data Management Systems

	Multistore (Single Interface)	Polystore (Multiple Interfaces)
Loosely coupled	**PolyBase** BigIntegrator FORWARD Apache Drill (Calcite) QoX QUEPA Odyssey	Myria
Tightly coupled	**RHEEM** MuSQLE HadoopDB	**ESTOCADA** Polypheny-DB
Hybrid	**CloudMdsQL** SparkSQL	**BigDAWG**

RHEEM – Plan Enumeration

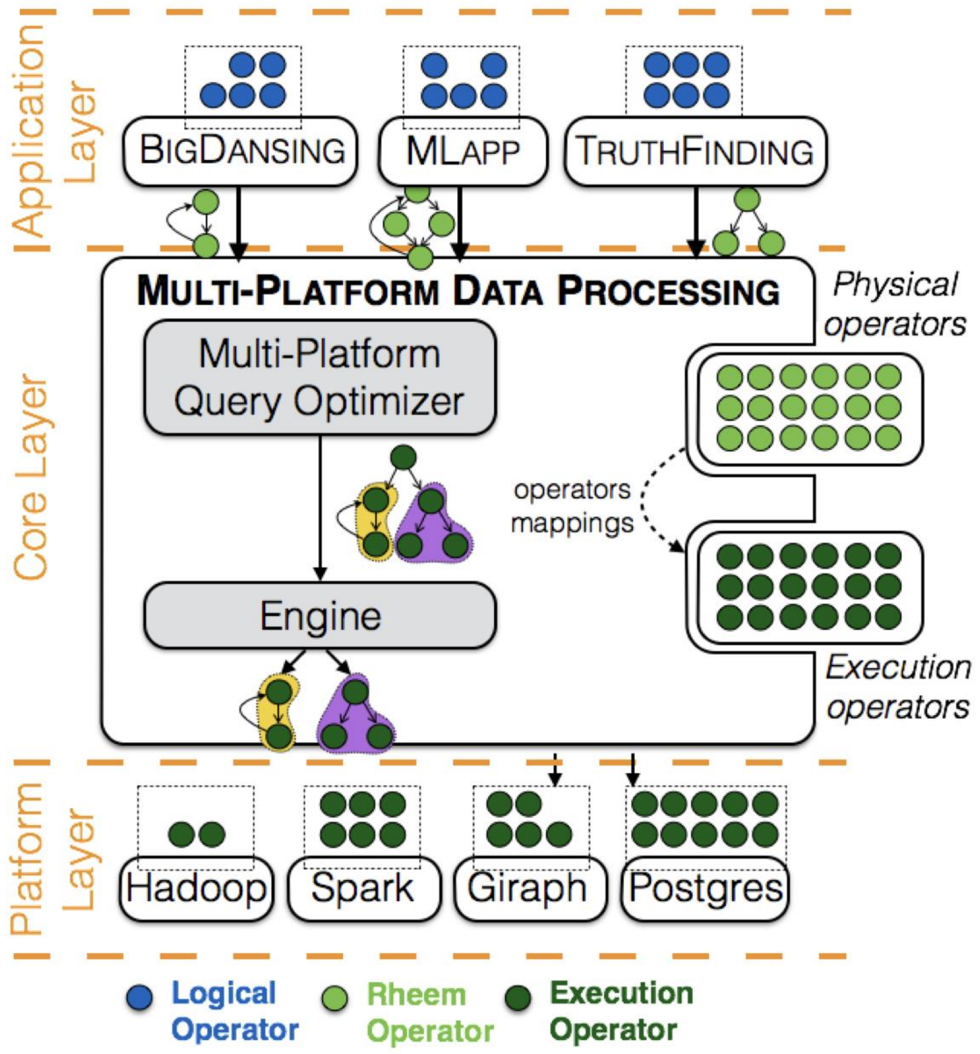
- **Input: Directed RHEEM dataflow plan**
  - RheemLatin DSL
  - RheemStudio
  - Java, Scala, Python
  - REST Endpoint
- **Output: Inflated operator plan with migration steps between platforms**
  - Map fix RHEEM operator to execution platform
  - Apply mappings between single logical operators to n* execution operators
  - Resolve **minimum** conversion tree to transfer data between multiple platforms



Kruse et al., RHEEMix in the Data Jungle–A Cross-Platform Query Optimizer, VLDB J., 2020.

RHEEM

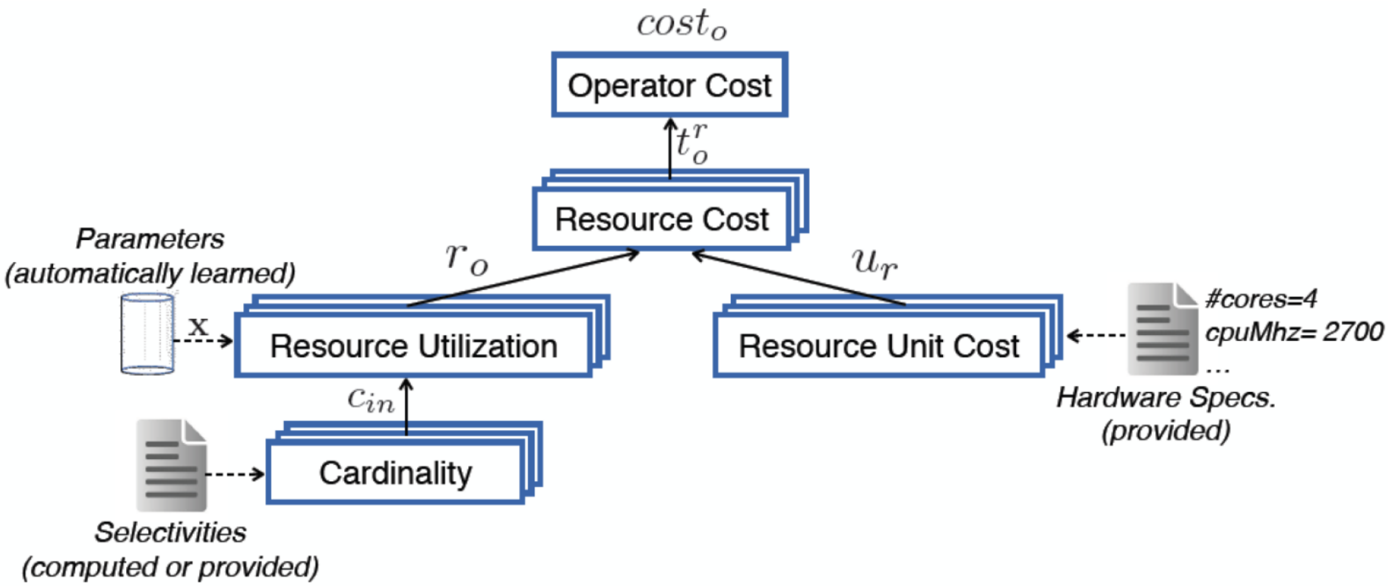
- Decoupling application task from (multi-platform) execution
- Mapping of *platform-agnostic* operator to *platform-specific* operators using **LAV**
- Resolve Migrations using **Channel Conversion Graph**
- Supports
  - InMemory (Java), GraphChi
  - PostgreSQL
  - Flink, Spark
- Developed as **Apache Wayang (Incubating)**



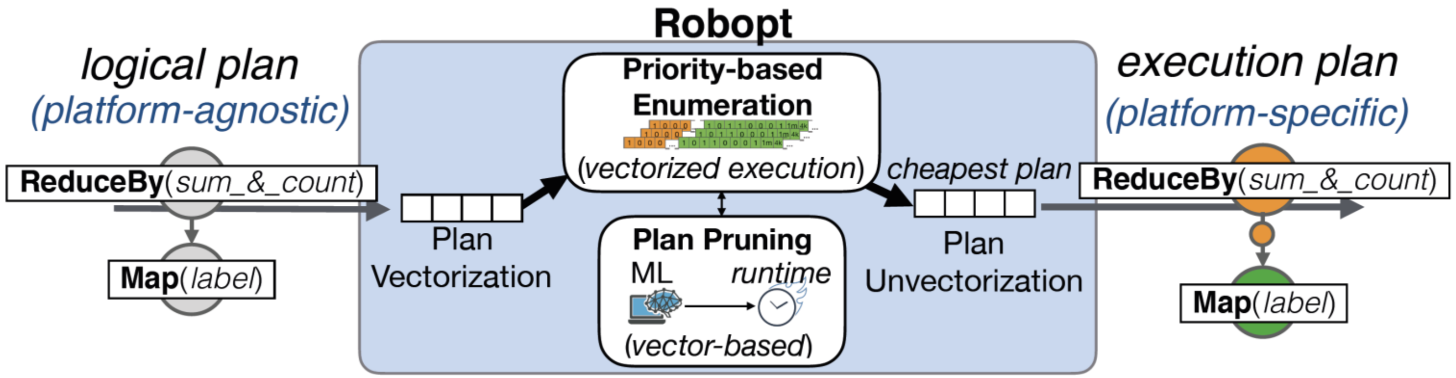
Agrawal et al., Rheem: Enabling multi-platform task execution, SIGMOD, 2018.

RHEEM – Plan Optimization

- **First version:** *genetic* cost-model learner, loss reduction
  - operator execution costs
  - Samples cardinalities and reduce size estimation function e.g., Filter: *card (Filter)=* $\mathbf{c_{in}}$*(Filter) **$\boldsymbol{\sigma_f}$ *for selectivity* ***f***
- **ML version (Robopt):** *supervised* fine-level cost-tuning
  - Encodes *logical operator-, platforms* and *movements* into vectors
  - Vectorized execution plan
  - ML-model selects enumerated vector plans with platform-agnostic operations
  - **Optimizes** the **order** of executing RHEEM operators



Kruse et al., RHEEMix in the Data Jungle–A Cross-Platform Query Optimizer, VLDB J., 2020.



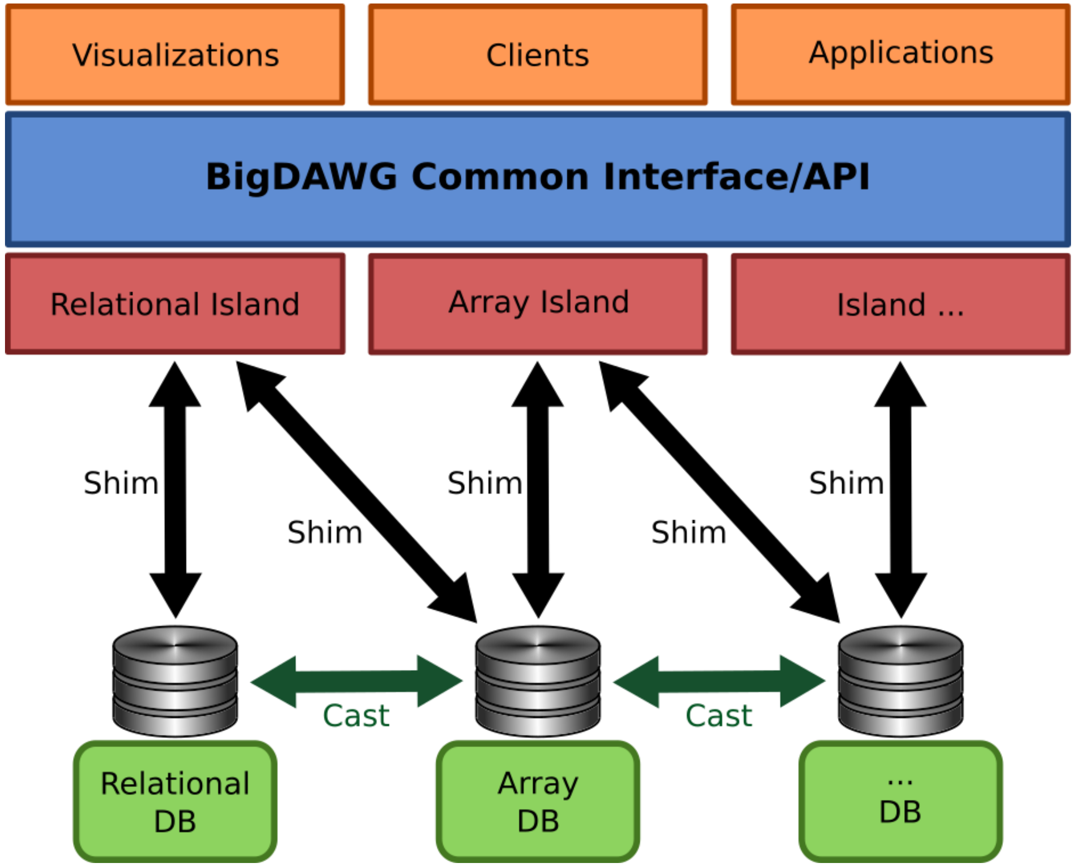
Kaoudi et al., ML-based cross-platform query optimization, ICDE, 2020.

Overview: Polyglot Data Management Systems

	Multistore (Single Interface)	Polystore (Multiple Interfaces)
Loosely coupled	**PolyBase** BigIntegrator FORWARD Apache Drill (Calcite) QoX QUEPA Odyssey	Myria
Tightly coupled	**RHEEM** MuSQLE HadoopDB	**ESTOCADA** Polypheny-DB
Hybrid	**CloudMdsQL** SparkSQL	**BigDAWG**

BigDAWG – Overview and Architecture

- **Developed at:**
  - **At:** Intel Science and Technology Center for Big Data (MIT)
  - **Between:** 2015 and 2019
- **Use Cases:**
  - Medical applications (MIMIC II)
  - Ocean Metagenomic Analysis

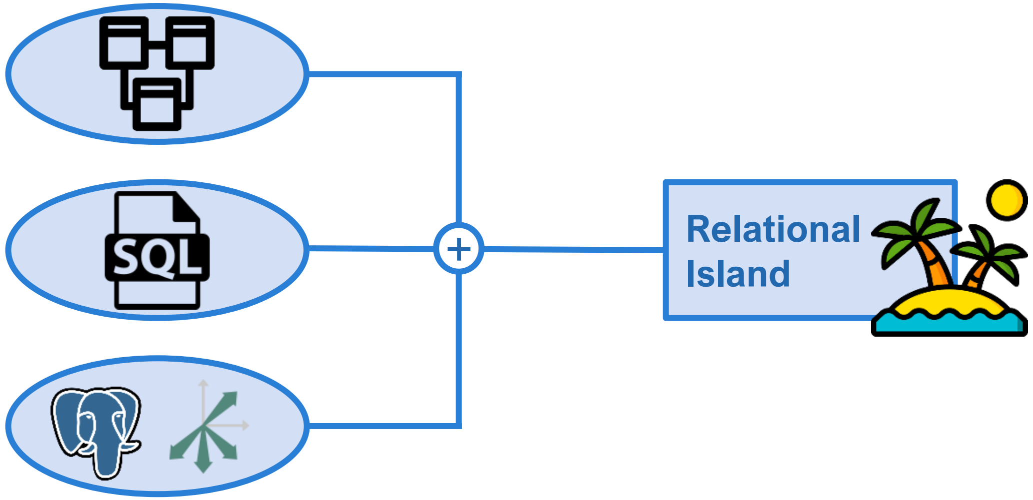


BigDAWG Polystore

Architecture figure: Gadepally et al., The BigDAWG polystore system and architecture, IEEE HPEC, 2016.

BigDAWG – Islands of Information

- 3 components of virtual islands:
  - Data model
  - Query language
  - Storage engines
- Degenerate islands to achieve semantic completeness
- **Shims**: semantical mapping between island and data store
- **Casts** and **Scope**: accessing multiple islands
- Extensible by implementing new islands



Array scope

Cast operation

Relational scope

```
bdarray(
    filter(
        bdcast(
            bdrel(SELECT val FROM table),
            postgres_results,
            ´<val:double> [i=0:*,100,0]´,
            array),
        val < 35)
)
```

Example from: O’Brien, Polystore Systems for Complex Data Management, IEEE HPEC, 2017.

BigDAWG – Performance Profiling

- Training mode:
  - all plans of a query are executed
  - the best is stored in the preference matrix

- Optimized mode:
  - either the best plan from the preference matrix
  - or a random plan is executed

- Opportunistic mode:
  - Similar to optimized mode
  - Additional evaluations during times of low system utilization
  - Additional evaluations if new stores become available

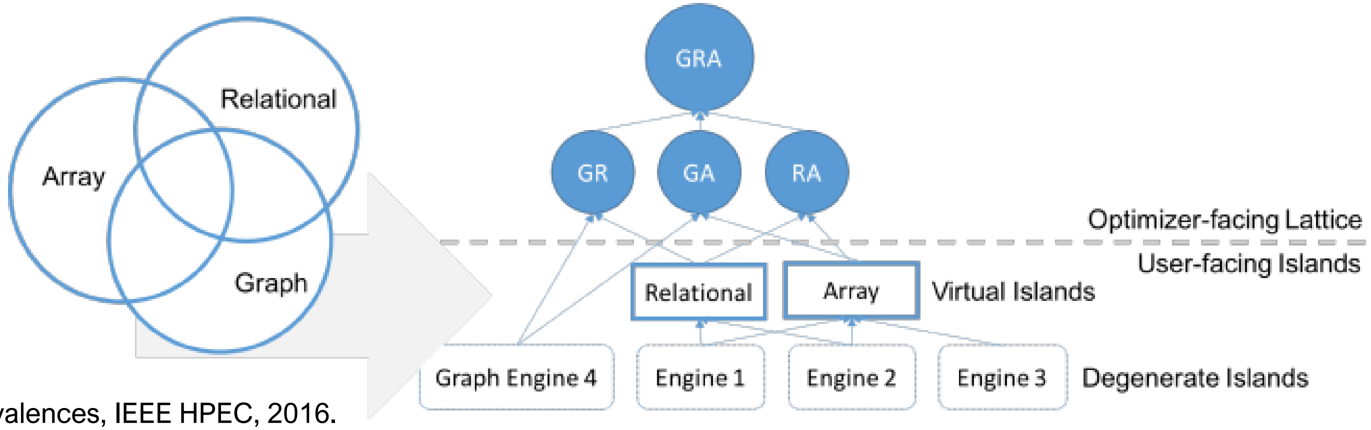
BigDAWG – Semantic Equivalence

- „Semantically equivalent queries […] are substitutable“
- Encode intersecting sets of semantic capabilities using a **semantic lattice**
- Capture semantic equivalent (sub-)queries in a semantic dictionary (Equivalence Rule)
- 3 types of semantic containment:
  - Order of result entries
  - Expressivness of semantics
  - Backward compatibility for primitive types

```
AFL: multiply(a, b)
```
Multiply SciDB

```
SQL: SELECT a.row_num, b.col_num,
            SUM(a.value*b.value)
     FROM a, b
     WHERE a.col_num = b.row_num
     GROUP BY a.row_num,
b.col_num;
```
Multiply Relational Database

```
{AFL:multiply(a,b);
 SQL:aggregate(join(a,b));
 „All values for (int64,
integer)“}
```
Equivalence Rule



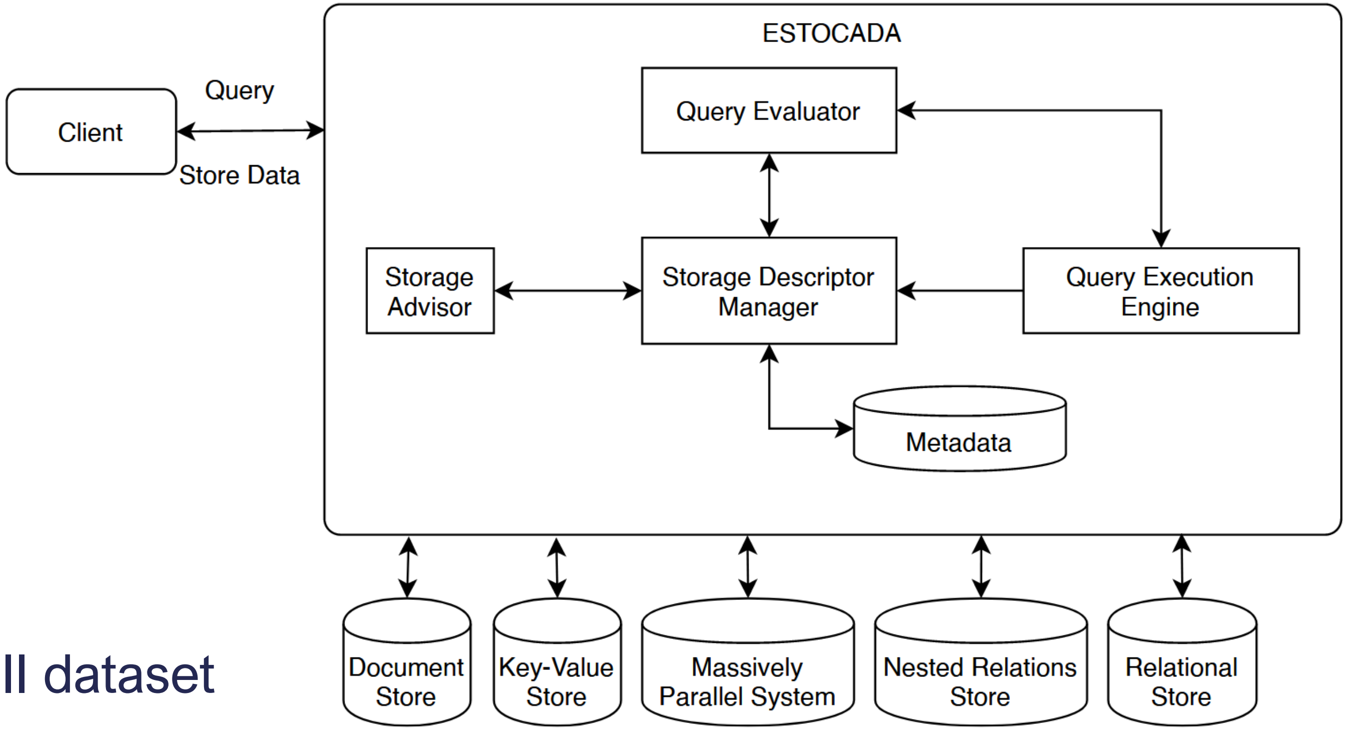
Figures from: She et al., BigDAWG Polystore Query Optimization Through Semantic Equivalences, IEEE HPEC, 2016.

Overview: Polyglot Data Management Systems

	Multistore (Single Interface)	Polystore (Multiple Interfaces)
Loosely coupled	**PolyBase** BigIntegrator FORWARD Apache Drill (Calcite) QoX QUEPA Odyssey	Myria
Tightly coupled	**RHEEM** MuSQLE HadoopDB	**ESTOCADA** Polypheny-DB
Hybrid	**CloudMdsQL** SparkSQL	**BigDAWG**

ESTOCADA

- Developed by University of San Diego and INRIA*
- Focus on view-based Query Rewriting (Local-as-view)
- Leveraging possible data redundancy and previously computed query results for improving performance
- Can be built into existing Polystores (e.g., BigDAWG, SparkSQL, Tatooine)
- Functional demonstration based on MIMIC III dataset



* Institut national de recherche en sciences et technologies du numérique

Architecture figure: Bugiotti et al., Invisible Glue: Scalable Self-Tuning Multi-Stores, CIDR, 2015.

ESTOCADA – Virtual Views

- Relational model as pivot model
- Virtual views on underlying models (encoded *relationally*)
- Differences in semantics modeled by integrity contraints
  - tuple-generating dependencies
  - equality-generating dependencies
- Encodings/Models hidden (only necessary for query rewriting)

$Collection_J(name, id)$	Collection membership
$Child_J(parentId, childId, key, type)$	JSON tree structure
$Eq_J(x, y)$	Value Equalitiy semantics
$Value_J(x, y)$	Value assignment

$$Collection_J(n, x) \wedge Collection_J(n, y) \rightarrow x = y \quad (1)$$

$$Child_J(p, c_1, k, t) \wedge Child_J(p, c_2, k, t) \rightarrow c_1 = c_2 \quad (2)$$

$$Eq_J(x, y) \rightarrow Eq_J(y, x) \quad (3)$$

$$Eq_J(x, y) \wedge Eq_J(y, z) \rightarrow Eq_J(x, z) \quad (4)$$

$$Eq_J(p, p') \wedge Child_J(p, c, k, t) \rightarrow \exists c' \; Eq_J(c, c') \wedge Child_J(p', c', k, t) \quad (5)$$

$$Value_J(i, v_1) \wedge Value_J(i, v_2) \rightarrow v_1 = v_2 \quad (6)$$

Example: Alotaibi et al., Towards Scalable Hybrid Stores: Constraint-Based Rewriting to the Rescue, SIGMOD, 2019.

ESTOCADA – Query Language and Rewriting

QBT^XM:

- Block-based integration language
- Each block contains native query language
  - FOR clause: Bind variables from stores
  - WHERE clause: Selections on bound variables
  - RETURN clause: Construct new data based on variable bindings

AsterixDB query

```
FOR AJ:{SELECT  M.patientID AS patientID,
                A.admissionID AS admissionID,
                A.report AS report
        FROM  MIMIC M, M.admissions A}
RETURN SJ:{"patientID":patientID,
           "admissionID":admissionID,
           "report":report}
```

SOLR result model

Query Rewriting:

- Optimized version of PACB algorithm
- Query rewriting using all virtual as well as materialized views

Logical Query Plan:

- Translation of PACB result into logical plan
  - Subqueries and supported operators pushed down to stores
  - Handling of unsupported operators and cross-store-joins by the integration layer

Example: Alotaibi et al., ESTOCADA: Towards Scalable Polystore Systems, PVLDB, 2020.

Overview: Polyglot Data Management Systems

	Multistore (Single Interface)	Polystore (Multiple Interfaces)
Loosely coupled	**PolyBase** BigIntegrator FORWARD Apache Drill (Calcite) QoX QUEPA Odyssey	Myria
Tightly coupled	**RHEEM** MuSQLE HadoopDB	**ESTOCADA** Polypheny-DB
Hybrid	**CloudMdsQL** SparkSQL	**BigDAWG**

CloudMdsQL

- Functional SQL-like language implemented in LeanXcale (Research system)
- Multistore with (current) support for
  - PostgresQL
  - MongoDB
  - Apache Spark
  - *(Python)*
- Abtraction layer for data retrieval
  - Preserves the semantics of the underlying data stores
  - A query may contain embedded (native) subqueries
  - Python functions to query API-only query interfaces
- Mediator/wrapper architecture
- Relational model as internal data model

```
T1(x int, y int)@rdb = ( SELECT x, y FROM A )
T2(x int, z array)@mongo = {*
db.B.find( {$lt: {x, 10}}, {x:1, z:1, _id:0} )
*}
SELECT T1.x, T2.z
FROM T1, T2
WHERE T1.x = T2.x AND T1.y <= 3
```

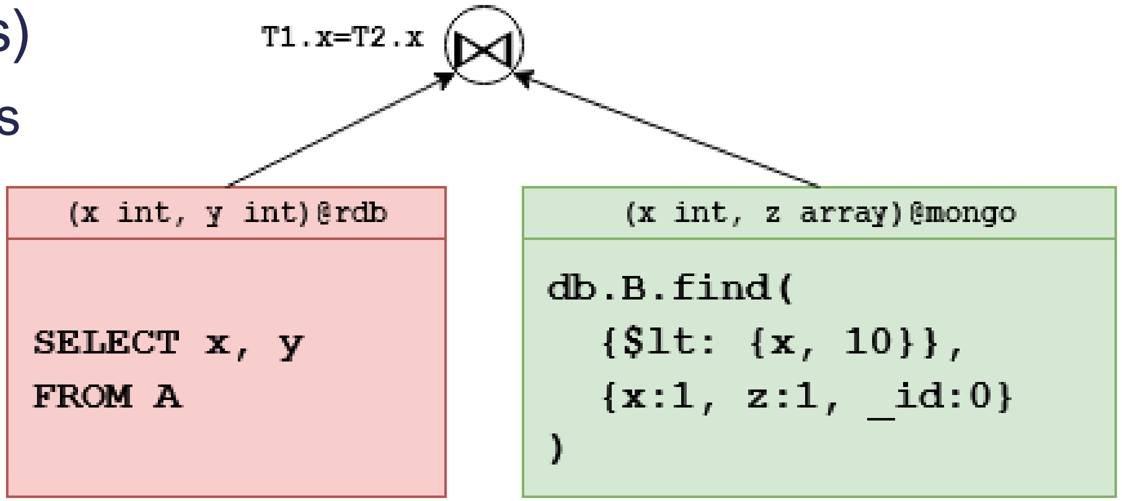
Code Example: Kolev et al., The CloudMdsQL Multistore System, SIGMOD, 2016.

CloudMdsQL – Query Execution

- Queries usually consist of subqueries and an integration SELECT statement

```
T1(x int, y int)@rdb = ( SELECT x, y FROM A )
T2(x int, z array)@mongo = {*
db.B.find( {$lt: {x, 10}}, {x:1, z:1, _id:0} )
*}
SELECT T1.x, T2.z
FROM T1, T2
WHERE T1.x = T2.x
```

- The system creates query execution plans (QEPs)
  - Subqueries are pushed down to the wrappers/stores
  - Subquery results are transformed into a relational format
  - Relational data is combined using Bind Joins



Code Example: Kolev et al., The CloudMdsQL Multistore System, SIGMOD, 2016.

CloudMdsQL – Query Optimization

- The optimization search space for consists of all query rewritings by
  - Pushing down select operations
  - Expressing Bind Joins
  - Join ordering
- Search space is small, thus a simple exhaustive search strategy is used
- Usage of a simple catalog for comparing rewritten queries:
  - Data collection cardinalities
  - Indexes
  - Attribute selectivities
  - Simple cost models
- Local cost models provided by probing and sampling by the wrappers

Wrap Up: Polyglot Data Management Systems

Automatic cross-platform optimization, operator placement & data migration

Multistore

Sing
Interfa

Data virtualization, single T-SQL interface, data fabric (Microsoft)

PolyBase

BigIntegrator

Virtual views, constrained based data transformation & block-based integration language

Application-driven, customizable data islands & semantic equivalence

cc d

Odyssey

RHEEM

MuSQLE

HadoopDB

ESTOCADA

Polypheny-DB

Extensible pivot query language, data store agnostic (LeanXcale)

CloudMdsQL

SparkSQL

BigDAWG

Part V

Open Challenges

Open Challenges: Overview

Unified Access vs. Unique Features
How to design a suitable interface?

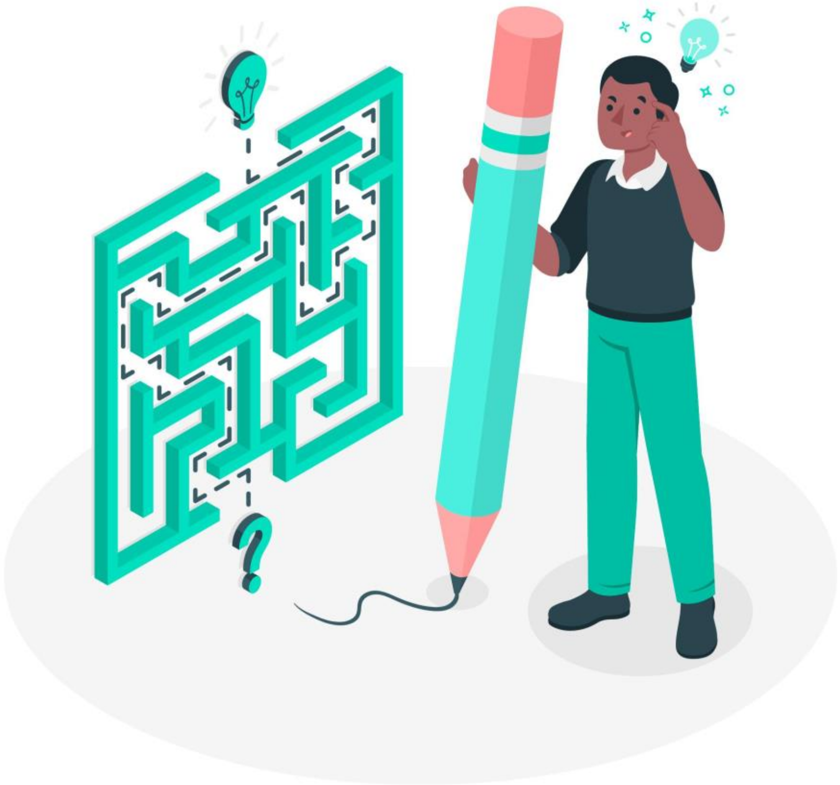
Ad Hoc Data Manipulation
How to push user updates to the stores?

Adaptive Reconfiguration
How to react to changing requirements?

Cross-System Query Optimization
How to find the optimal query plan?

Streaming & Real-Time Readiness
How to integrate real time requirements?

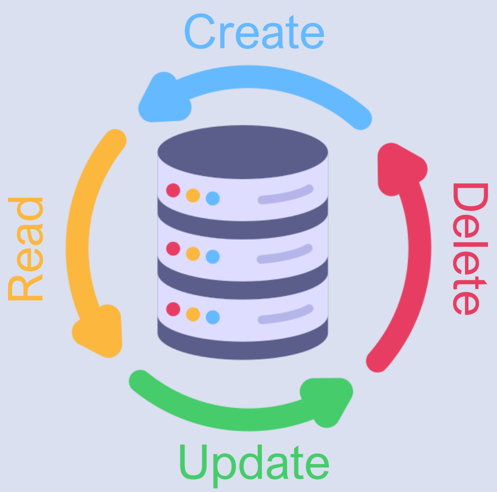
Multi-Model Schema Management
How to update schema mappings?



Open Challenges: (i) Unified Access vs. Unique Features

„smallest common denominator“

- ✓ Simple to build
- ϟ Not very powerful
- ϟ Loss of semantic/functional features



Mediator query language with embedded store query languages

- ✓ Easily extensible
- ✓ Full semantic/functional complexity
- ϟ Does not hide complexity
- ϟ Prevents intra store optimization potential

```
T1(x int, y int)@rdb = ( SELECT x, y FROM A )
T2(x int, z array)@mongo = {*
db.B.find( {$lt: {x, 10}}, {x:1, z:1, _id:0} )
*}
SELECT T1.x, T2.z
FROM T1, T2
WHERE T1.x = T2.x
```

CloudMdsQL

All-powerful query language

- ✓ Hidden complexity
- ✓ Full semantic/functional complexity
- ϟ Super complex to build
- ϟ Extensibility challenging
- ϟ Feasible?

```
FROM readings AS r
GROUP BY r.gas AS g
SELECT ELEMENT {
  gas: g,
  count: count(group),
  avg: avg(
     FROM group AS p
     SELECT ELEMENT p.r.num)}
```

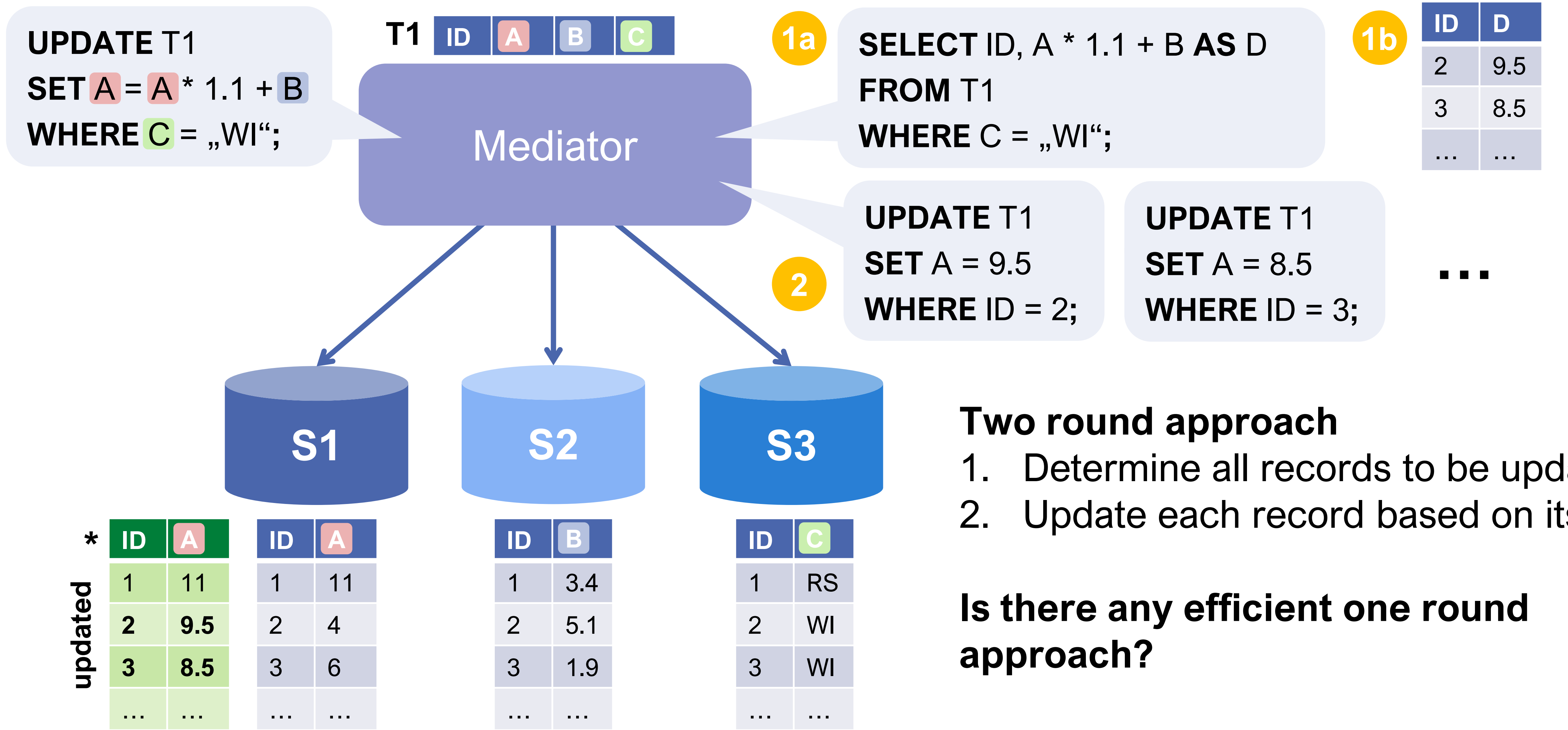
SQL++

```
@eq{
  complex        :    yes,
  null_eq_null   :    null,
  null_and_true  :    null
} (r=r)
```

SQL++

Code Examples: Kolev et al., The CloudMdsQL Multistore System. SIGMOD, 2016.
Ong et al., The SQL++ Semi-structured Data Model and Query Language. arxiv.org, 2014.

Open Challenges: (ii) Ad Hoc Data Manipulation

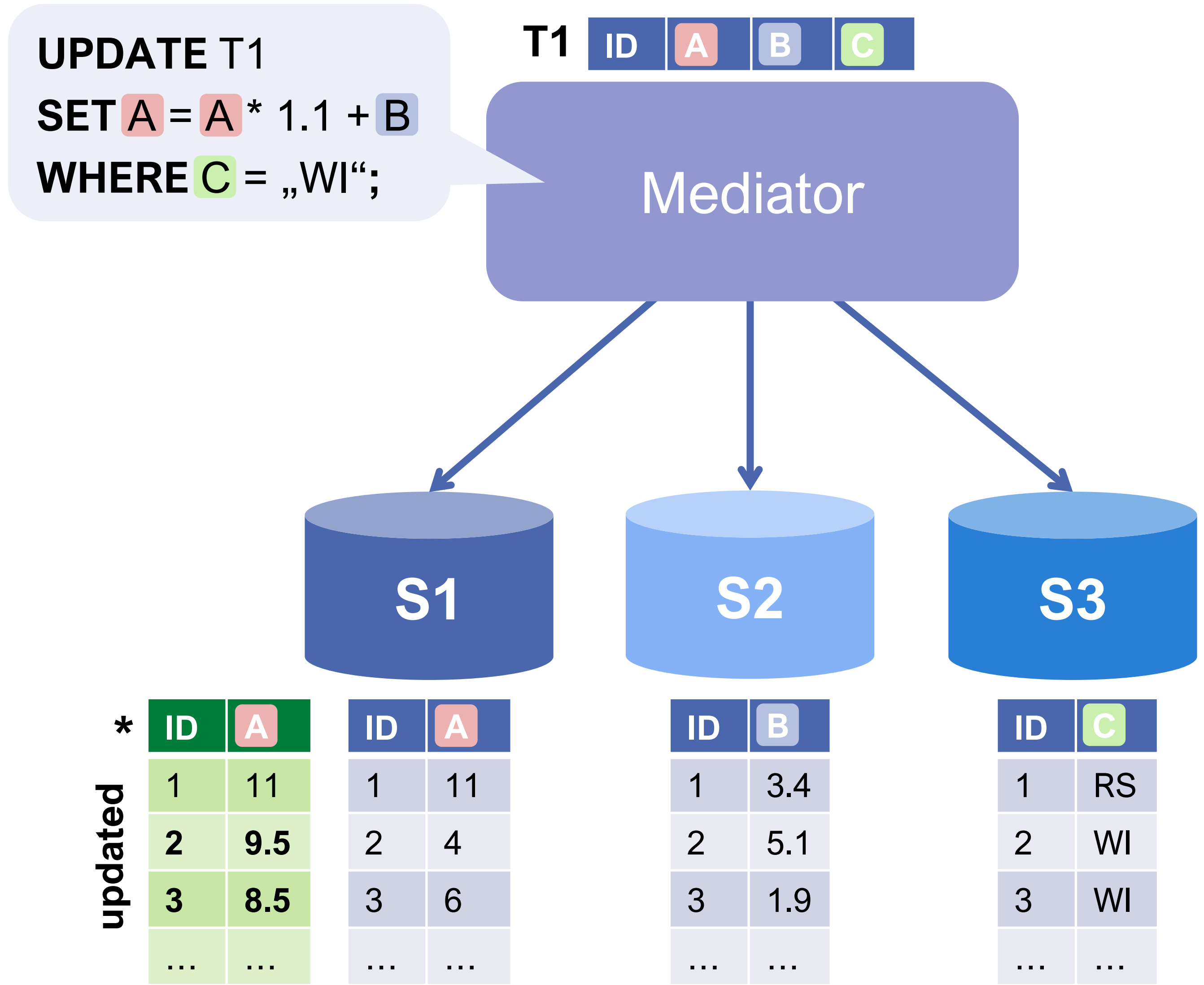


Two round approach

1. Determine all records to be updated
2. Update each record based on its ID

Is there any efficient one round approach?

Open Challenges: (ii) Ad Hoc Data Manipulation



Further Challenges:

How to ensure cross-store

- **Atomicity**, **Isolation** & **Durability**
  - logging
  - locking
  - recovery
- **Consistency**
  - check constraints
  - referential integrity

if individual stores do not support such mechanisms?

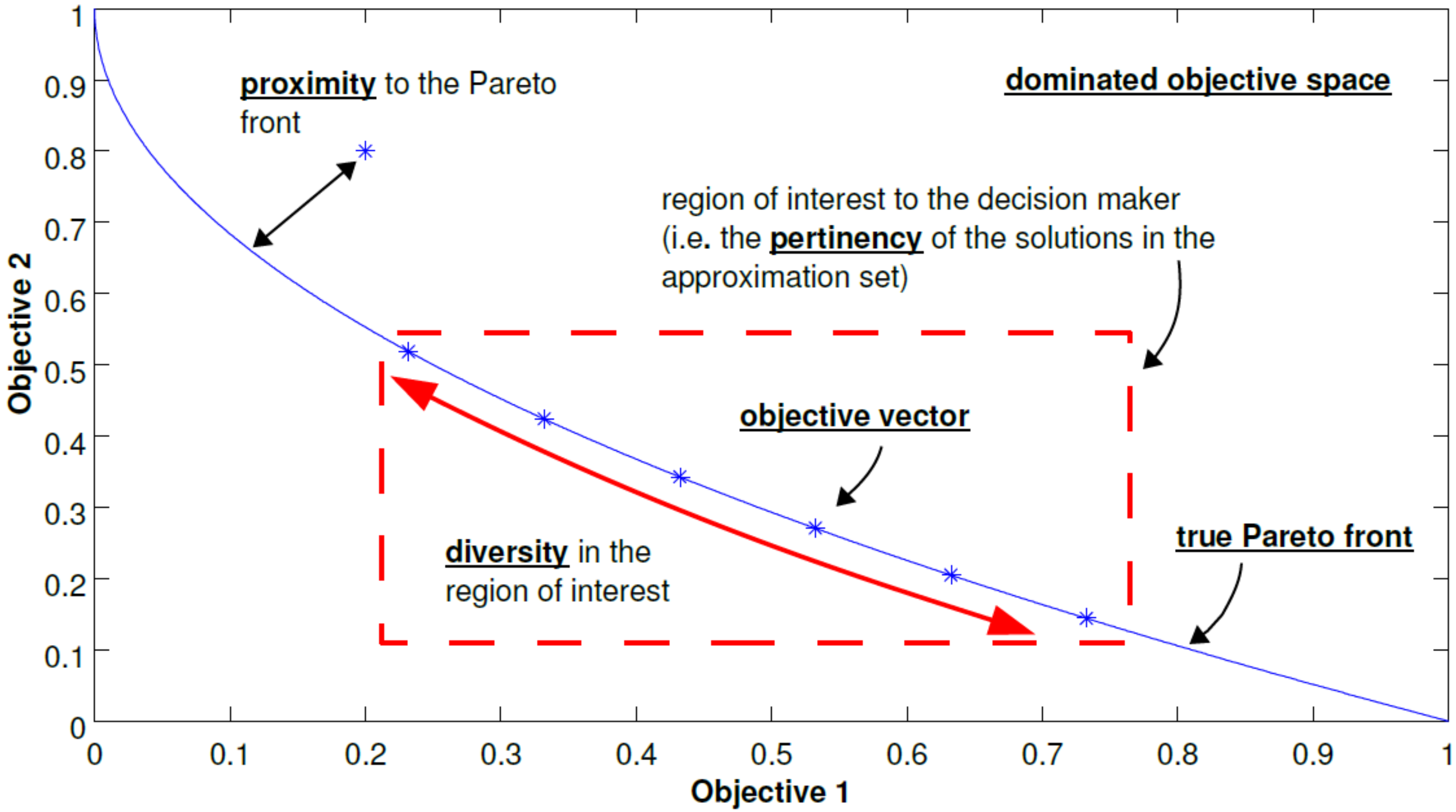
Open Challenges: (iii) Adaptive Reconfiguration

- Detecting changing requirements or workloads?
  - Fluctuating traffic throughout the day
  - Singular events (e.g. Black Friday)
  - Additional users in a multi-tenant environment
- Adapting/reconfiguring the system
  - Adding or removing resources
  - Reorganization (e.g. splitting a hot range)
- Changing the system topology
  - Data migration between stores
    (e.g. write-heavy data to main-memory database)

at runtime

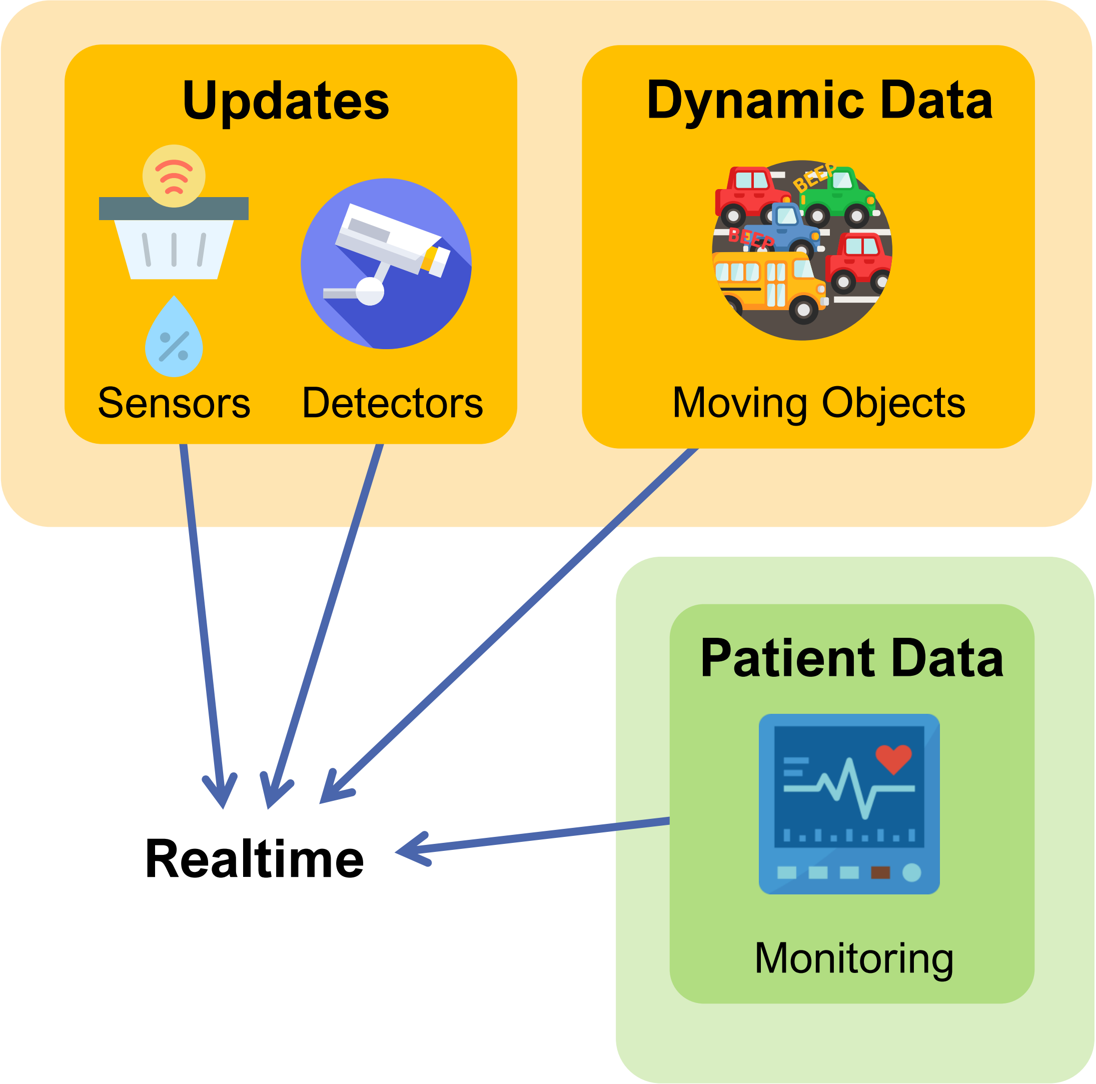
Open Challenges: (iv) Cross-System Query Optimization

- Operator Placement
  - Data vs. Operator Shipping
  - Migration Paths
- Pareto Optimum of Objectives
  - Latency
  - Throughput
  - Planning
  - Application Objective
- ML-based optimization
  - Hard constraint for query correctness in optimization
  - Join-Ordering for sub-query

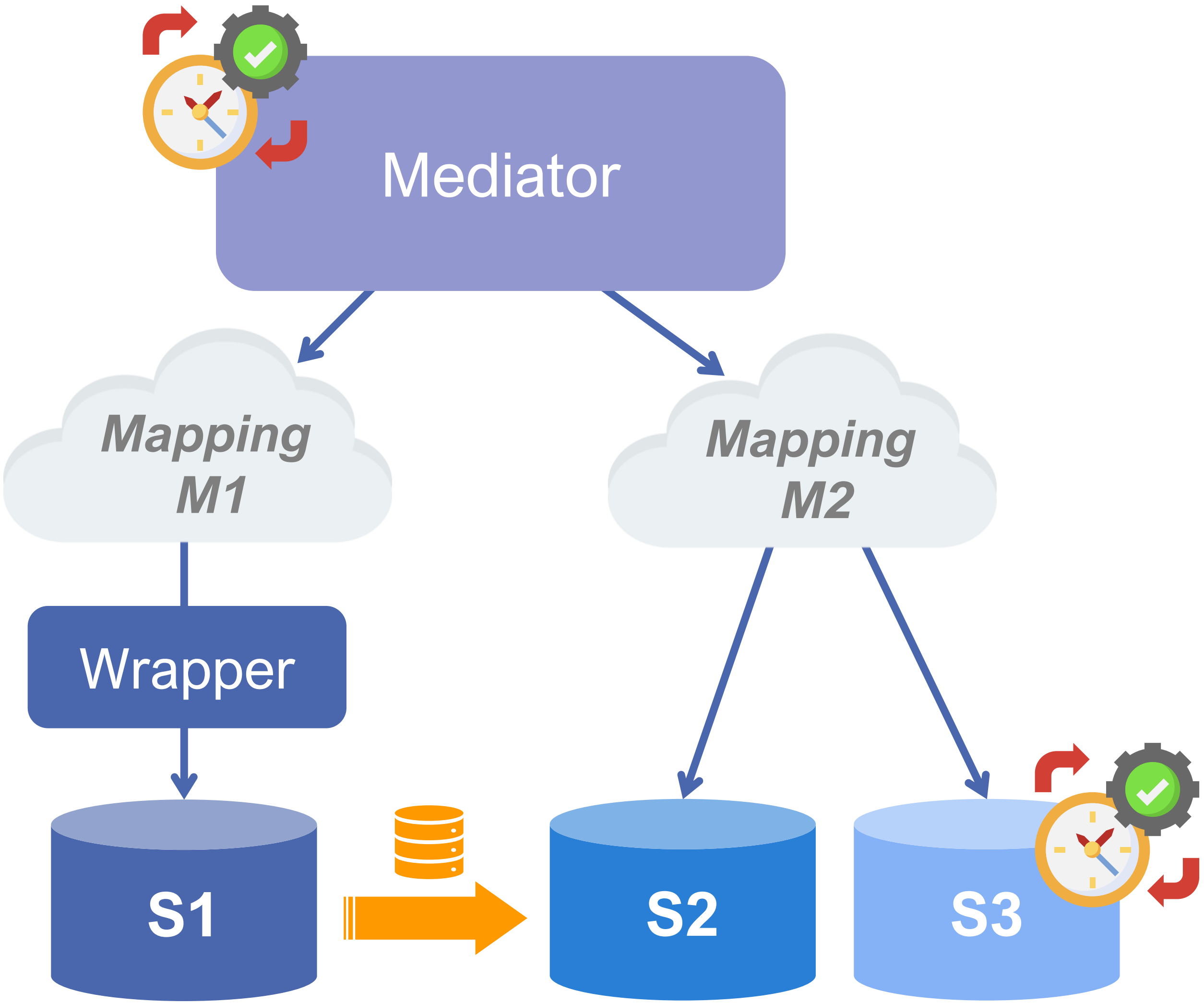


Open Challenges: (v) Streaming & Real-Time Readiness

- Streaming Workloads
  - Expose streaming capabilities
  - Integrate streaming with storage systems
- Push-based features
  - Triggers, ECA rules
  - Change notifications
- Caching
  - Materialized views
  - Cache coherence / cache invalidation



Open Challenges: (vi) Multi-Model Schema Management



- Mappings between global & local Schemas
  - fundamental for query rewriting
  - cross-model (e.g., SQL ↔ graph)
  - via wrapper
- Update of Mappings
  - Evolution of global schema
  - Evolution of local schema
  - Migration of data between stores
- Composition/Extraction of Mappings for data migration

Further Readings

1

Towards Polyglot Data Stores

Overview and Open Research Questions

DANIEL GLAKE*, FELIX KIEHN*, and MAREIKE SCHMIDT*, Universität Hamburg
FABIAN PANSE and NORBERT RITTER, Universität Hamburg

Nowadays, data-intensive applications face the problem of handling heterogeneous data with sometimes mutually exclusive use cases and soft non-functional goals such as consistency and availability. Since no single platform copes everything, various stores (RDBMS, NewSQL, NoSQL) for different workloads and use-cases have been developed. However, since each store is only a specialization, this motivates progress in polyglot data management emerged new systems called Mult- and Polystores. They are trying to access different stores transparently and combine their capabilities to achieve one or multiple given use-cases. This paper describes representative real-world use cases for data-intensive applications (OLTP and OLAP). It derives a set of requirements for polyglot data stores. Subsequently, we discuss the properties of selected Multi- and Polystores and evaluate them based on given needs illustrated by three common application use cases. We classify them into functional features, query processing technique, architecture and adaptivity and reveal a lack of capabilities, especially in changing conditions tightly integration. Finally, we outline the benefits and drawbacks of the surveyed systems and propose future research directions and current challenges in this area.

CCS Concepts: • **Information systems → DBMS engine architectures.**

Additional Key Words and Phrases: polyglot persistence, multi-/polystore, data management, adaptivity, query processing.

2

Enabling Query Processing across Heterogeneous Data Models: A Survey

Ran Tan, Rada Chirkova
Department of Computer Science
North Carolina State University
Raleigh, North Carolina
Email: rtan2@ncsu.edu, rychirko@ncsu.edu

Vijay Gadepally
Lincoln Laboratory
Massachusetts Institute of Technology
Lexington, Massachusetts
Email: vijayg@ll.mit.edu

Timothy G. Mattson
Intel Corporation
Portland, Oregon
Email: timothy.g.mattson@intel.com

Abstract*—Modern applications often need to manage and analyze widely diverse datasets that span multiple data models [1], [2], [3], [4], [5]. Warehousing the data through Extract-Transform-Load (ETL) processes can be expensive in such scenarios. Transforming disparate data into a single data model may degrade performance. Further, curating diverse datasets and maintaining the pipeline can prove to be labor intensive. As a result, an emerging trend is to shift the focus to federating specialized data stores and enabling query processing across heterogeneous data models [6]. This shift can bring many advantages: First, systems can natively leverage multiple data models, which can translate to maximizing the semantic expressiveness of underlying interfaces and leveraging the internal processing capabilities of component data stores. Second, federated architectures support query-specific data integration with just-in-time transformation and migration, which has the potential to significantly reduce the operational complexity and overhead. Projects that focus on developing systems in this research area stem from various backgrounds and address diverse concerns, which could make it difficult to form a consistent view of the work in this area. In this survey, we introduce a taxonomy for describing the state of the art and propose a systematic evaluation framework conducive to understanding of query-processing characteristics in the relevant systems. We use the framework to assess four representative implementations: BigDAWG [7], [8], CloudMdsQL [9], [10], Myria [11], [12], and Apache Drill [13].

***Keywords*-Cross-model query processing; Query-specific data integration; Taxonomy; Evaluation framew**

I. INTRODUCTION

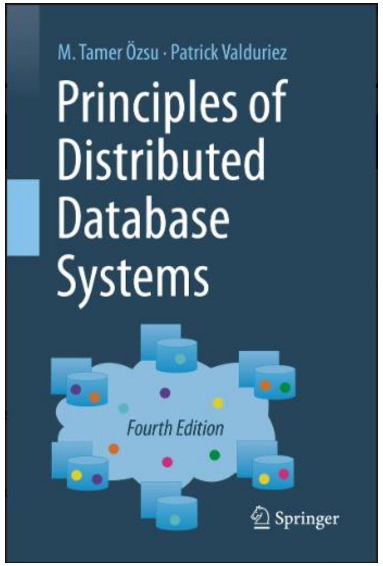
Modern applications often need to ma widely diverse datasets that span multi

events expressed as JSON (JavaScript Object Notation) documents, social-media data recorded via key-value pairs, and weather feeds stored in relational tuples to predict traffic flows. Finally, in data journalism [5], journalists work with Tweet texts, relational databases provided by governments and institutions, and RDF-formatted Linked Open Data to support content management for writing political articles.

In these and other scenarios, warehousing the data using Extract-Transform-Load (ETL) processes can be very expensive. First, transforming disparate data into a single chosen data model may degrade performance. Indeed, there appears to be no "one size fits all" solution for all markets [14], [15], as specialized models and architectures enjoy overwhelming advantages in data warehousing, text searching, stream processing, and scientific databases. Second, curating diverse datasets and maintaining the pipeline could turn out to be labor intensive [16]. One major reason is that rules and functions in ETL scripts do not adapt to changes in data and analytical requirements, and changes in application logic often result in the modification of ETL scripts.

For these and other reasons, a number of projects are shifting the focus to federating specialized data stores and enabling query processing across heterogeneous data models [6]. This shift can bring many advantages. First, the

3



4

CIKM 2018

Multi-model Databases and Tightly Integrated Polystores

Current Practices, Comparisons, and Open Challenges

UNIVERSITY OF HELSINKI
FACULTY OF MATHEMATICS AND PHYSICS
Charles University
UNIVERSITÉ PARIS SUD

Jiaheng Lu, Irena Holubová, Bogdan Cautis

1 https://arxiv.org/pdf/2204.05779.pdf

2 https://par.nsf.gov/servlets/purl/10074262

3 https://link.springer.com/book/10.1007/978-3-030-26253-2

4 https://www.cs.helsinki.fi/u/jilu/documents/CIKMTutorial2018.pdf

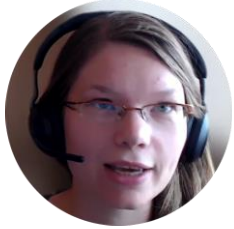
Thanks …

VLDB 2022

Questions?

Slides available at:
vldb2022.dbis.hamburg


Felix Kiehn


Mareike Schmidt


Daniel Glake


Fabian Panse


Wolfram Wingerath


Benjamin Wollmer


Martin Poppinga


Norbert Ritter