Slides credit: Matthias Boehm

Data Integration and Large Scale Analysis

03 - Replication, MoM, and Enterprise Applications Integration

Shafaq Siddiqi

Graz University of Technology, Austria

Agenda

- Overview of Group Project
- Motivation and Terminology
- Distributed TX & Replication Techniques
- Asynchronous Messaging
- Message-oriented Integration Platforms

Overview Programming Projects & Exercises

Overview Group Project

- Example: Entity Matching

ID	Name	Address	City	Type
a1	arnie mortons of chicago	435 s. la cienega blv.	los angeles	american
a2	arts delicatessen	12224 ventura blvd.	studio city	american
a3	fenix	8358 sunset blvd. west	hollywood	american
a4	restaurant katsu	1972 n. hillhurst ave.	los angeles	asian

Table A: Restaurants in Data Source A

ID	Name	Address	City	Type
b1	arnie mortons of chicago	435 s. la cienega blvd.	los angeles	steakhouses
b2	arts deli	12224 ventura blvd.	studio city	delis
b3	fenix at the argyle	8358 sunset blvd. w.	hollywood	french (new)
b4	katsu	1972 hillhurst ave.	los feliz	japanese

Table B: Restaurants in Data Source B

Overview Group Project

- Team
  - **1-3 person teams** (w/ clearly separated responsibilities)
  - Exercise description on the TeachCenter

ID	Title	Author	Journal	Year
a1	Deep Learning	I. Goodfellow	JMLR	2016
a2	Graph Neural Networks	T. Kipf	IEEE Access	2017
a3	Support Vector Machines	C. Cortes	Machine Learnin	1995
a4	Semantic Similarity	M. Li	Comput. Linguist.	2003

Table A: Papers in Data Source A

bD	Title	Author	Journal	Year
b1	Deep Learning	I. Goodfellow	JMLR	2016
b2	Graph Neural Networks	T. Kipf	IEEE Access	2016
b3	Support Vector Machines	C. Cortes	Mach. Learn.	1995
b4	Semantic Similarity	M. Li	Comput. Linguist.	2002

Table B: Papers in Data Source B

Group Project

- **Description:** You are given two bibliographic datasets (DBLP1.csv, Scholar.csv) and a ground-truth mapping (DBLP-Scholar_perfectMapping.csv). The goal is to build an end-to-end pipeline to identify matching records referring to the same publication.
  (https://data.dws.informatik.uni-mannheim.de/benchmarkmatchingtasks/index.html)

[Task 01]: Entity matching (40 Points)

1.1 Data Preparation (10 pts)

- Clean and normalize text; tokenize title, extract author last names, parse year
- Deduplicate using Jaccard ≥ 0.95 or relaxed rule

Report: number of input records, duplicate groups, final unique records

1.2 Blocking (10 pts)

- Apply ≥2 blocking rules (e.g., title prefix, author overlap)
- Generate union of matches; report candidate pairs count, and % gold retained

1.3 Similarity Scoring (10 pts)

- Compute cosine similarity (TF-IDF)
- Explore Jaccard, Dice, Levenshtein, or field-specific similarity scores

Report: number of pairs with cosine ≥ 0.95 + comparison summary

1.4 Matching & Evaluation (10 pts)

- Evaluate against perfect mapping

Report: P/R/F1 at thresholds: 0.70, 0.80, 0.90, 0.95

Group Project

Task 02: Feature Vector and ML Model (50 Points)

Train a Machine Learning model to classify whether a candidate pair refers to the same entity.

2.1 Create a Training Dataset

Start from the output pairs generated in Task 1 (e.g., pairs_scored.csv, which contains all candidate pairs and their similarity scores).

- Assign label = 1 if the (DBLP, Scholar) pair exists in the gold mapping
- Label = 0 otherwise

Hint: You could use DBLP-Scholar_perfectMapping.csv to identify true matches.

2.2 Feature Engineering

- Compute at least 6 features using existing similarity scores (title_cos, title_jacc, auth_olap, year_ok, venue_exact)

Hint: For better training data introduce additional features such as:

- Title length difference
- Jaccard/Dice similarity (authors or venue)
- Shared token count or character overlap
- Levenshtein distance
- Cosine similarity on other fields

2.3 Preprocess Training Data

- Handle missing values and duplicates
- Normalize or scale numeric features
- Address class imbalance using one of:
  - Undersampling
  - Class weights or
  - SMOTE

Group Project

2.4 Model Training

- Train both **Random Forest** and **SVM**
- Use **3-fold cross-validation**
- Report **Precision, Recall, and F1-score**

2.5 Evaluation

- Beat the baselines:
  - **SVM: F1 ≈ 0.76**
  - **RF: F1 ≈ 0.79**
- Discuss model performance and most useful features

Task 03: Reporting & Reproducibility (10 Points)

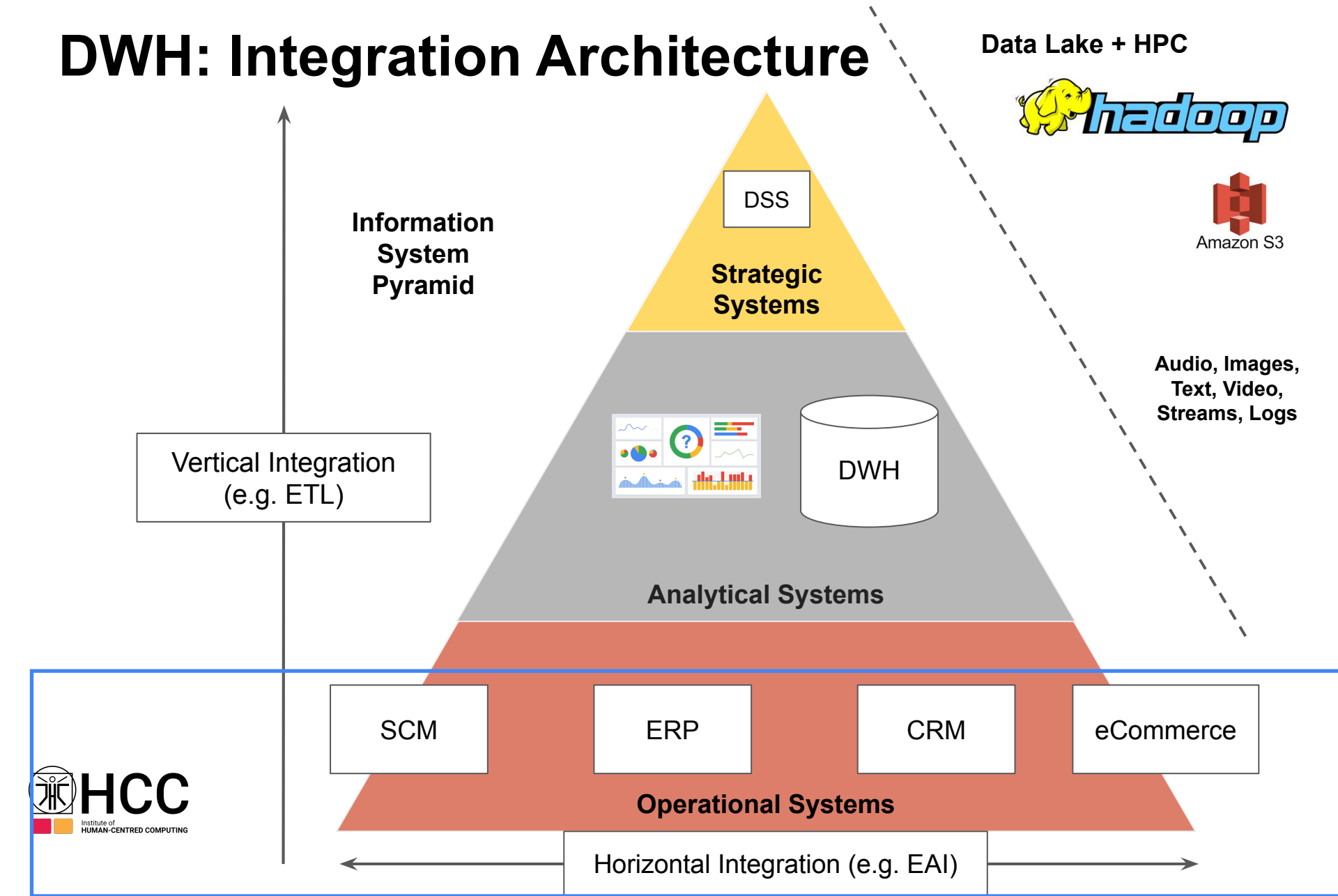
- Submit a precise and clear **report (PDF or Markdown)** should include:
  - Overview of Tasks 1, 2, 3
  - Tables with summaries:
    - Blocking results
    - Similarity scores
    - ML evaluation (P/R/F1)
  - Feature engineering descriptions
- Submission must include:
  - All Python/Jupyter code
  - requirements.txt
  - README.md with run instructions

- **Submission Deadline: January 10, 2026**

Motivation and Terminology

Replication, MoM, and EAI

DWH: Integration Architecture



Messaging

- Message
  - Piece of information in **certain structure**
  - Send fr**om source (transmitter)** over channel **to destination (receiver)**
  - **Syntax:** different **message formats** (binary, text, XML, JSON, Protobuf)
  - **Semantic:** different **domain-specific message schemas** (aka data models)

Sintax 1	**Sintax 2**
{ "temperature": { "value": 20, “unit": "Celsius", "timestamp": "2025-10-16T10:00:00Z" } }	<observation> <temperature> <value>20</value> <unit>Celsius</unit> <timestamp>2025-10-16T10:00:00Z</timestamp> </temperature> </observation>

Syntax 1 = Syntax 2?

Messaging

- Message
  - Piece of information in **certain structure**
  - Send fr**om source (transmitter)** over channel **to destination (receiver)**
  - **Syntax:** different **message formats** (binary, text, XML, JSON, Protobuf)
  - **Semantic:** different **domain-specific message schemas** (aka data models)

JSON	**XML**
{ "temperature": { "value": 20, "unit": "Celsius", "timestamp": "2025-10-16T10:00:00Z" } }	<observation> <temperature> <value>20</value> <unit>Celsius</unit> <timestamp>2025-10-16T10:00:00Z</timestamp> </temperature> </observation>

Syntax 1 = Syntax 2? **No!**

Messaging

- Message
  - Piece of information in **certain structure**
  - Send fr**om source (transmitter)** over channel **to destination (receiver)**
  - **Syntax:** different **message formats** (binary, text, XML, JSON, Protobuf)
  - **Semantic:** different **domain-specific message schemas** (aka data models)

Semantic 1	**Semantic 2**
{ " ": { "value": 20, "timestamp": "2025-10-16T10:00:00Z" } }	{ " ": { "value": 20, "timestamp": "2025-10-16T10:00:00Z" } }

Semantic 1 = Semantic 2?

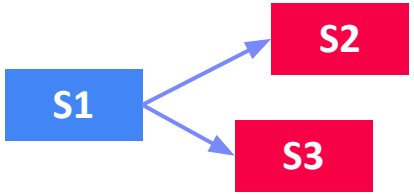
Messaging

- Message
  - Piece of information in **certain structure**
  - Send fr**om source (transmitter)** over channel **to destination (receiver)**
  - **Syntax:** different **message formats** (binary, text, XML, JSON, Protobuf)
  - **Semantic:** different **domain-specific message schemas** (aka data models)

Semantic 1	**Semantic 2**
{ "temperature": { "value": 20, “unit": "Celsius", "timestamp": "2025-10-16T10:00:00Z" } }	{ "voltage”: { "value": 20, “unit": "Volts", "timestamp": "2025-10-16T10:00:00Z" } }

Messaging

- Synchronous Messaging
  - **Strict consistency requirements**
  - Overhead for distributed transactions **via 2PC (Two phases commit)**
  - Low local autonomy



Messaging

- Asynchronous Messaging
  - **Loose coupling**, eventual consistency requirements
  - **Batching** for efficient replication and updates **(Replication)**
  - **Latency** of update propagation

Types of Data Formats

- General-Purpose Formats
  - **CLI/API** access to DBs, KV-stores, doc-stores, time series DBs, etc
  - **CSV** (comma separated values)
  - **JSON** (javascript object notation), **XML**

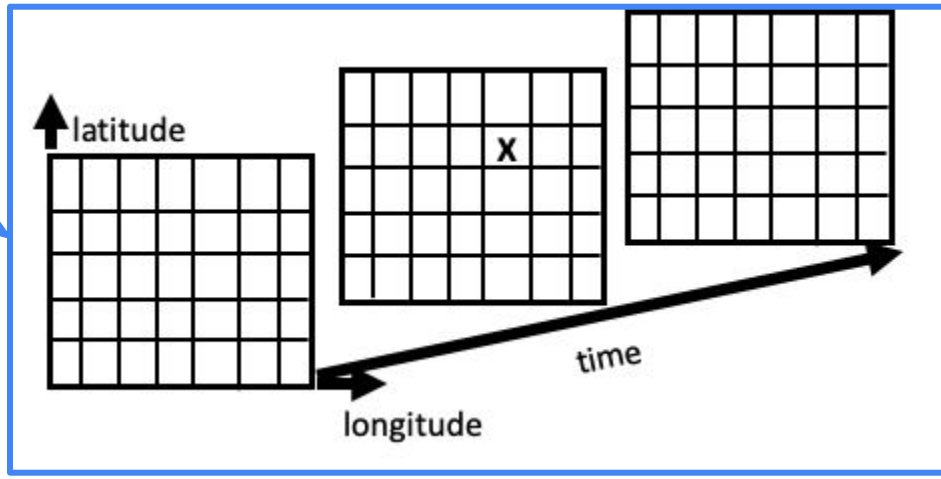
CLI: redis-cli GET temperature

API: GET /api/v1/data?metric=temperature

Types of Data Formats

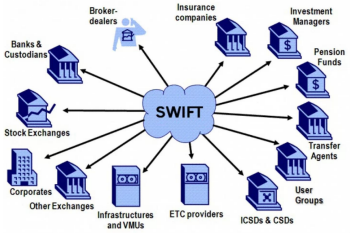
- Sparse Matrix Formats
  - **Matrix market:** text IJV (row, col, value)
  - Scientific formats: **NetCDF**, **HDF5**
- Others: Large-Scale Data Format
  - ORC (Optimized Row Column),
  - Parquet (column-oriented file formats)

```
%%MatrixMarket matrix coordinate real general
% ----------------------------------------------
% 0 or more comment lines
% ----------------------------------------------
5  5  8
1 1 1.000e+00
2 2 1.050e+01
3 3 1.500e-02
1 4 6.000e+00
4 2 2.505e+02
4 4 -2.800e+02
4 5 3.332e+01
5 5 1.200e+01
```



Example Domain-specific Message Formats

- Finance: SWIFT
  - Society for Worldwide Interbank Financial Telecommunication
  - >10,000 orgs (banks, stock exchanges, brokers and traders)
  - Network and message formats for financial messaging
  - MT and MX (XML, ISO 20022) messages



[https://ihodl.com]

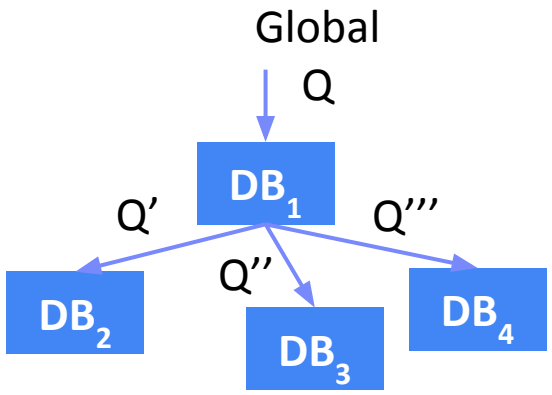
Types of Message-Oriented Middleware

- #1 Distributed TXs & Replication
- #2 Message Queueing
  - Persistent message queues with well-defined delivery semantics
  - Loose coupling of connected systems or services (e.g., availability)
- #3 Publish Subscribe
  - Large number of subscribers to messages of certain topics/predicates
  - Published messages forwarded to qualifying subscriptions
- #4 Integration Platforms
  - Inbound/outbound connectors for external systems (e.g. In: MQTT - Out: JSON)
  - Sync and async messaging, message transformations, enrichment

Distributed TX & Replication Techniques

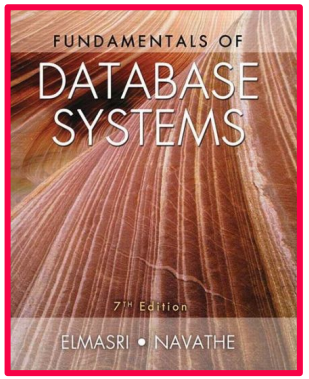
Distributed Database Systems

- Distributed DBS
  - Distributed database: Virtual (logical)
  - database that appears like a local database but consists of multiple physical databases
  - Multiple local DBMS, components for global query processing
  - **Terminology:** **virtual DBS** (homogeneous), **federated DBS** (heterogeneous)



VirtualDBS: same DBMS (e.g. PostgreSQL), same data model

FederatedDBS: different DBMS (e.g. MongoDB, MySQL)



Elmasri, R. (2008). Fundamentals of database systems. Pearson Education

Distributed Database Systems

- Terms
  - **Transparency:** user “perceives” the system as an unique database without knowing where data are located, underlying DBMS, and other details.
  - **Autonomy:** each local database can define their own schema, rules and control policies over their data. Nodes don't depend on a central coordinator
  - **Consistency:** keep data synchronized across all system nodes
  - **Efficiency/fault tolerance:** less communication overhead, more availability
- Tradeoffs
  - Transparency vs Autonomy
  - Consistency vs Efficiency/Fault Tolerance

Distributed Database Systems

- Challenges
  - **#1** Global view and unified query language --> schema architecture
  - **#2** Transparency → global catalog (which data, where are located, which DBMS, etc)
  - **#3** Distribution of data → data partitioning
  - **#4** Global queries → distributed join operators, etc (where to perform the join?)
  - **#5** Concurrent transactions → **2PC**
  - **#6** Consistency of copies → **replication**

Two-Phase Commit (2PC)

- Database Transaction
  - **Series of steps** that brings a database from a **consistent state** into another **consistent state**
  - **ACID properties**
    - **Atomicity:** A transaction is all or nothing: either all its operations are executed successfully, or none are.
    - **Consistency:** A transaction must take the database from one valid state to another, preserving all defined rules, constraints, and relationships.
    - **Isolation:** Transactions execute as if they were alone, meaning intermediate results are hidden from other concurrent transactions.
    - **Durability:** Once a transaction is committed, its effects are permanent, even if the system crashes right after.

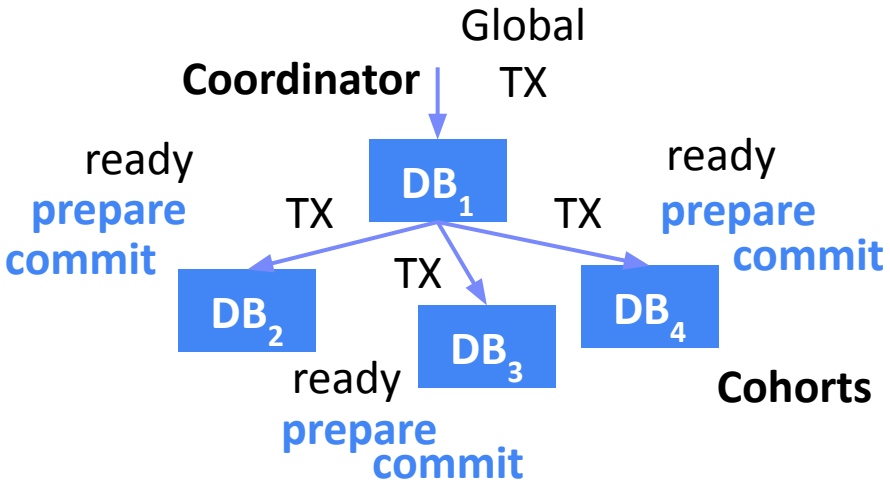
Two-Phase Commit (2PC)

- Problems in Distributed DBS
  - Node failures, and communication failures
  - **Distributed TX processing**
    - Distributed systems must coordinate updates across multiple nodes to maintain a consistent global view of the data (distributed transaction protocols)
    - **Atomicity?** All parts of a distributed transaction must either commit or abort together even if one node fails. **Two-Phase Commit (2PC) protocol.**
    - **Durability?** Once a transaction commits, its effects are permanent, even if a failure occurs afterward. **Recovery logs and backups.**

Two-Phase Commit (2PC)

Phase 1 PREPARE:

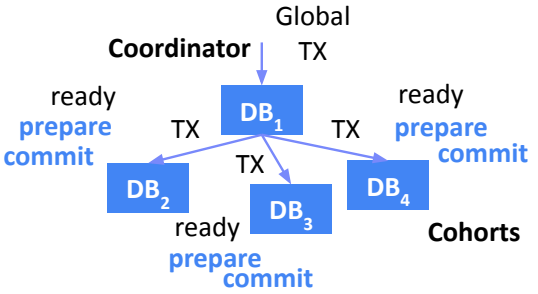
- The coordinator sends a **PREPARE** message to all participants.
- Each participant:
  - Execute the transaction locally (tentatively)
  - Writes all pending changes to its log (so they can be recovered later).
  - If everything succeeded, replies **VOTE COMMIT.**
  - If any problem occurred (e.g., constraint violation, crash), replies **VOTE ABORT.**



Two-Phase Commit (2PC)

Phase 2 COMMIT:

- Depending on the votes:
  - If all participants **voted COMMIT:**
    - The coordinator sends a **GLOBAL COMMIT** message.
  - Each node then:
    - Finalizes the transaction (makes changes permanent).
    - Releases locks and cleans up temporary data.
    - Writes a commit record in its log.
  - If any participant voted ABORT:
    - The coordinator sends a GLOBAL ABORT message.
    - Each node undoes its tentative changes and releases resources.



Two-Phase Commit (2PC)

Problems

During Phase 1 (PREPARE):

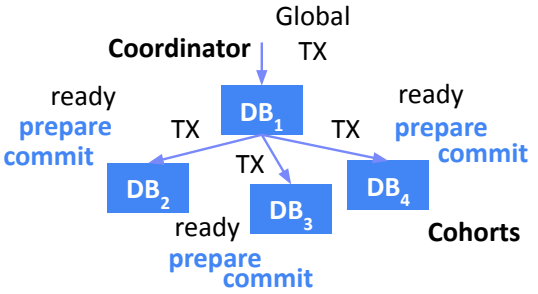
- If a node fails or reports an error, the coordinator cannot guarantee atomicity (all or nothing)
- The coordinator must decide GLOBAL ABORT (all participants abort the transaction)
- This avoids committing incomplete or inconsistent updates.

If the Coordinator Fails:

- Participants remain in an uncertain waiting state (they've voted but haven't received the final decision).
- When the coordinator recovers: it consults its log to decide whether to send COMMIT or ABORT.
- Until then, participants keep their locks (can **block** other transactions).

If the Network Fails:

- Messages between coordinator and some participants may be lost.
- The transaction remains in-doubt until communication is restored.
- Recovery procedures (timeouts, logs, coordinator election) are used to eventually reach consistency.



Extended Distributed Commit Protocols

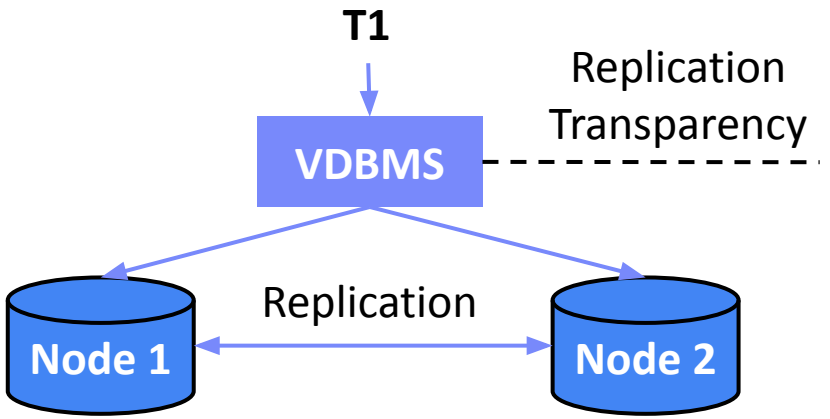
- 2PC Improvements
  - **Hierarchical Commit:** establish message tree from **coordinator** to **local nodes**
    - Message handling can occur in parallel within branches of the tree.
    - Reduces total communication time (especially when there are many participants).
    - Makes the protocol more scalable for large distributed systems.

Extended Distributed Commit Protocols

- 3PC (non-blocking version of 2PC)
  - a) CAN COMMIT?
    - Coordinator to all nodes
    - All YES (go to phase b)
    - If one replies NO → ABORT
  - b) PREPARE COMMIT
    - Coordinator to all nodes → Unanimous decision
    - Nodes now that TX will happens (they block their resources)
    - Nodes to coordinator → ACK
    - If coordinator fails: nodes can decide whether to perform the TX or not (Decision rules)
    - Decision rules: Find a new coordinator (another node in PRE-COMMIT)
    - YES (go to phase b) / NO (ABORT)
  - c) COMMIT?
    - Coordinator to all nodes →COMMIT
    - Write commit register and release blocks (Permanent TX)
    - Optional: Nodes to coordinator → ACK (For transaction logs in coordinator)

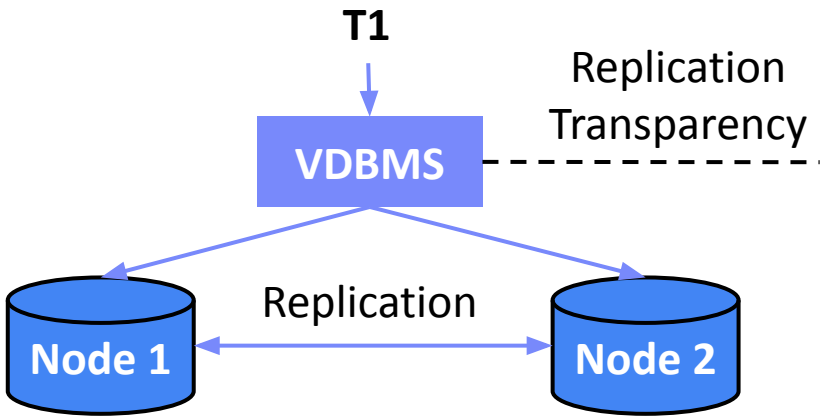
Replication Overview

- Replication
  - Redundancy of stored fragments
  - Availability/efficiency (read) vs update overhead / storage

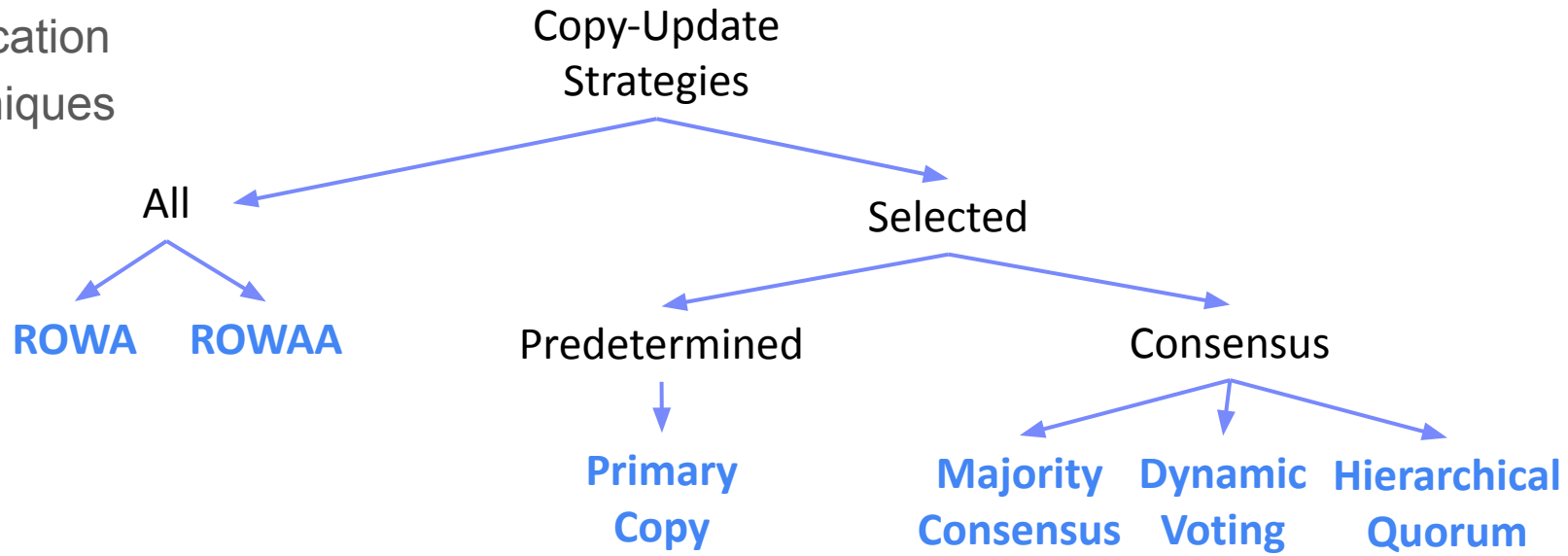


Replication Overview

- Replication
  - Redundancy of stored fragments
  - Availability/efficiency (read) vs update overhead / storage

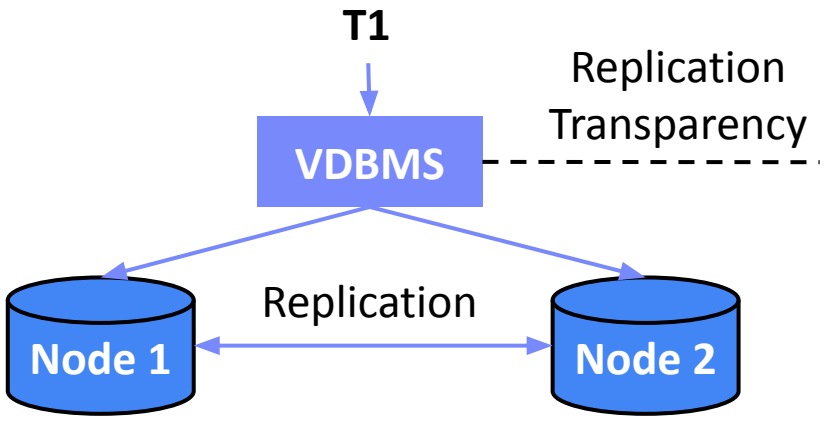


- Replication Techniques

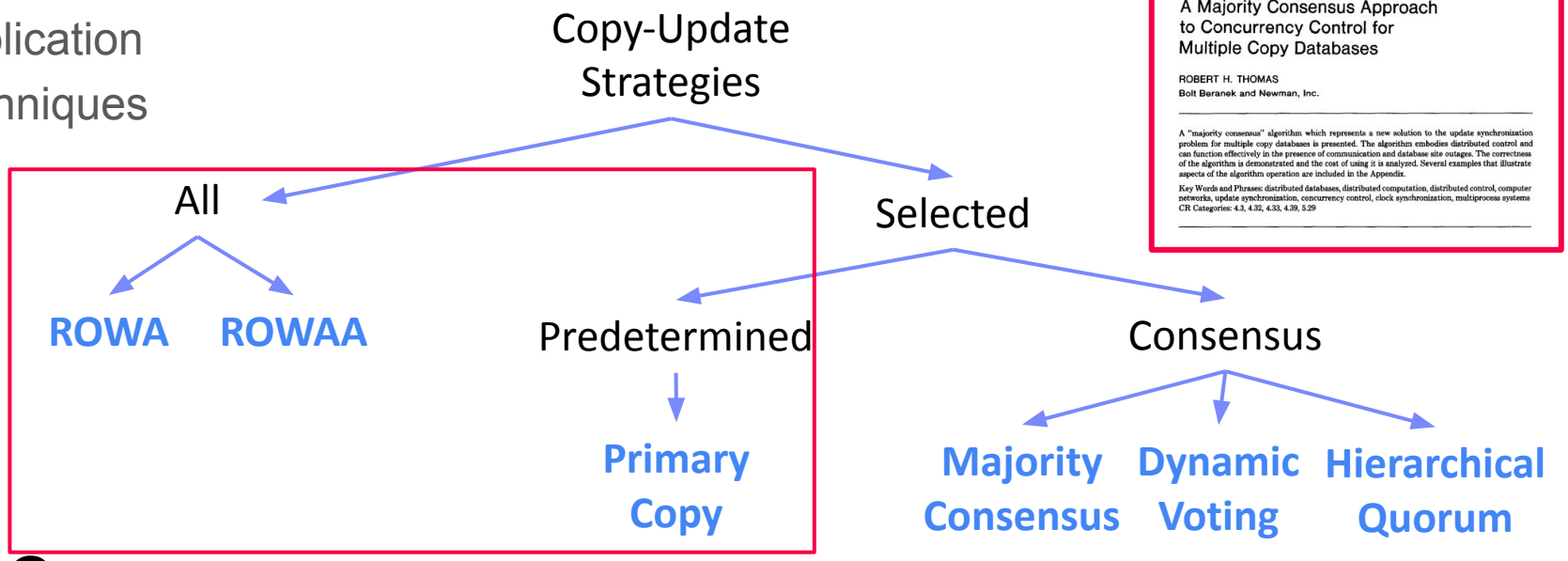


Replication Overview

- Replication
  - Redundancy of stored fragments
  - Availability/efficiency (read) vs update overhead / storage



- Replication Techniques



A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases

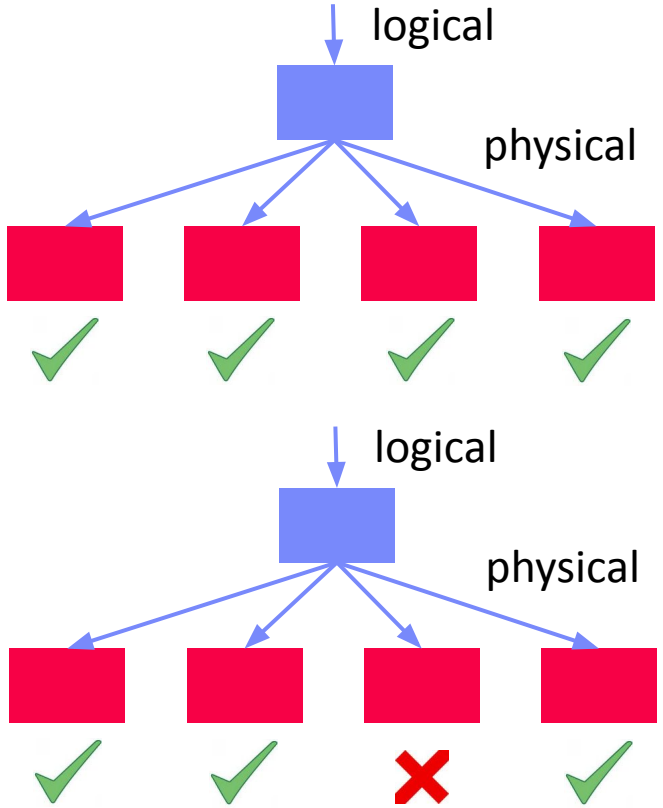
ROBERT H. THOMAS
Bolt Beranek and Newman, Inc.

A "majority consensus" algorithm which represents a new solution to the update synchronization problem for multiple copy databases is presented. The algorithm embodies distributed control and can function effectively in the presence of communication and database site outages. The correctness of the algorithm is demonstrated and the cost of using it is analyzed. Several examples that illustrate aspects of the algorithm operation are included in the Appendix.

Key Words and Phrases: distributed databases, distributed computation, distributed control, computer networks, update synchronization, concurrency control, clock synchronization, multiprocess systems
CR Categories: 4.3, 4.32, 4.33, 4.39, 5.29

Replication Techniques

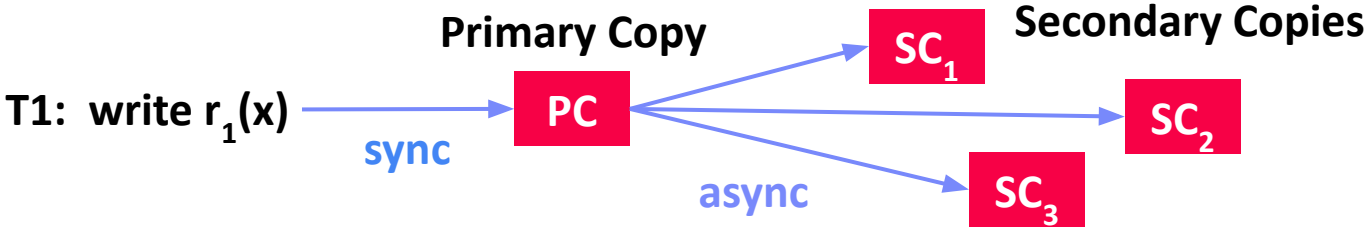
- ROWA
  - Read-One/Write-All
  - Read: good performance/availability
  - Write: high overhead and only successful if all available
- ROWAA
  - Read-One/Write-All-Available
  - Relaxed availability requirement for write operations



[**Jim Gray**, Pat Helland, Patrick E. O'Neil, Dennis Shasha: The Dangers of Replication and a Solution, **SIGMOD 1996**]

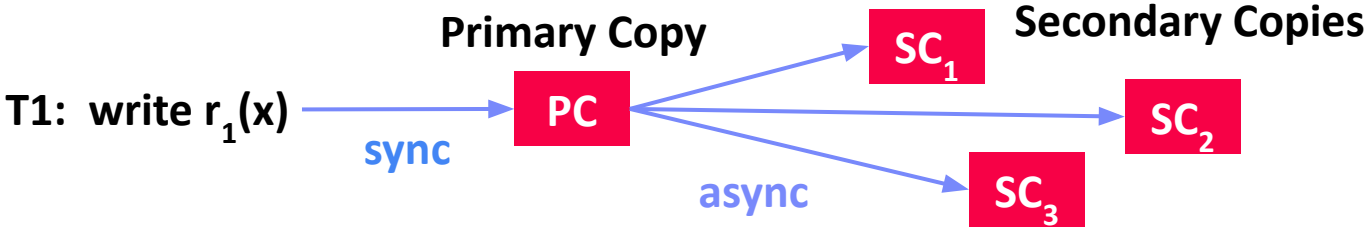
Replication Techniques, cont.

- Primary Copy
  - Update single primary copy **synchronously**
  - **Asynchronous propagation** of updates to other replicates, read from all



Replication Techniques, cont.

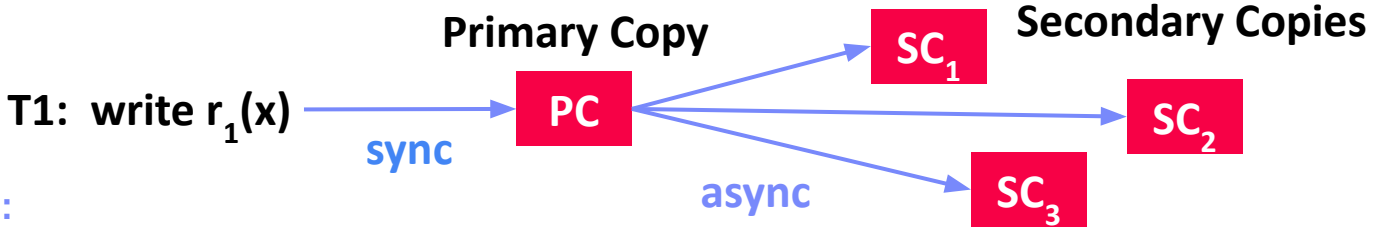
- Primary Copy
  - Update single primary copy **synchronously (write)**
  - **Asynchronous propagation** of updates to other replicates, read from all
  - Read from primary or **secondary (risk of stale data)**



  - **Pros:**
  - **Higher update performance** since only one primary replica handles writes, reducing synchronization overhead.
  - **Good locality** because updates can be executed close to the primary node, improving latency.
  - **High availability for reads** as read operations can be served from any replica.
  - **Supports load balancing** by distributing primary copies of different objects across multiple nodes.
  - **Efficient for read-heavy workloads** by minimizing coordination costs for frequent reads.

Replication Techniques, cont.

- Primary Copy
  - Update single primary copy **synchronously (write)**
  - **Asynchronous propagation** of updates to other replicates, read from all
  - Read from primary or **secondary (risk of stale data)**



  - **Cons:**
  - **Potentially stale reads:** secondary replicas may return outdated data due to asynchronous propagation.
  - **Single point of failure for writes:** if the primary copy fails, updates cannot proceed until recovery or re-election.
  - **Asynchronous propagation overhead:** requires mechanisms for log shipping, synchronization, and recovery.

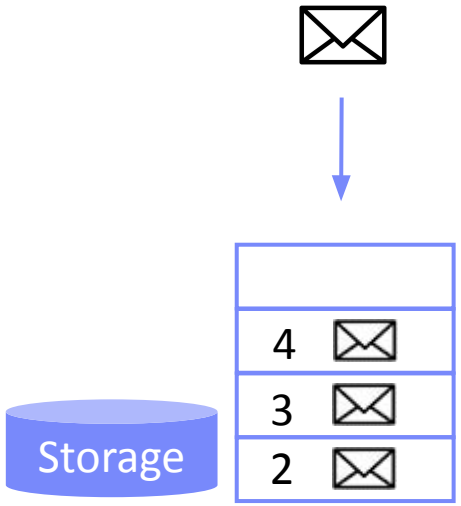
Asynchronous Messaging

Message Queueing

- Message
  - Atomic packet of data + meta data, wrapped as a message

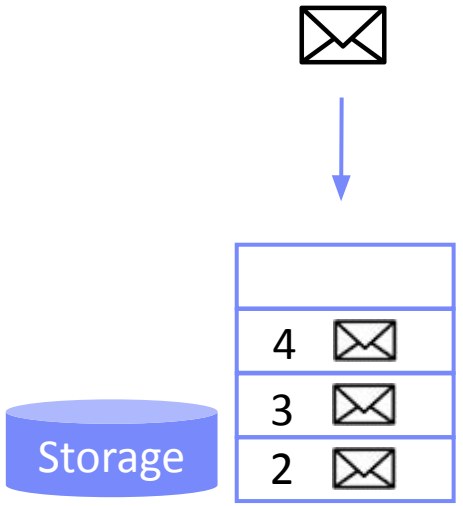
Message Queueing

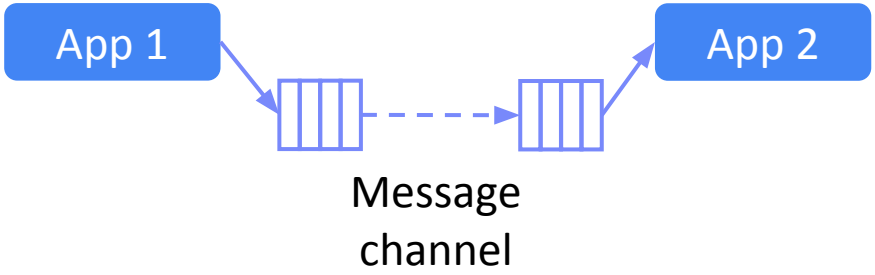
- Message
  - Atomic packet of data + meta data, wrapped as a message.
- Message Queue
  - FIFO or priority queue of messages.
  - In-memory, sometimes with persistent storage backend.



Message Queueing

- Message
  - Atomic packet of data + meta data, wrapped as a message
- Message Queue
  - FIFO or priority queue of messages
  - In-memory, sometimes with persistent storage backend
- Remote Message Queues
  - Loose coupling of applications
  - Independent of HW and OS



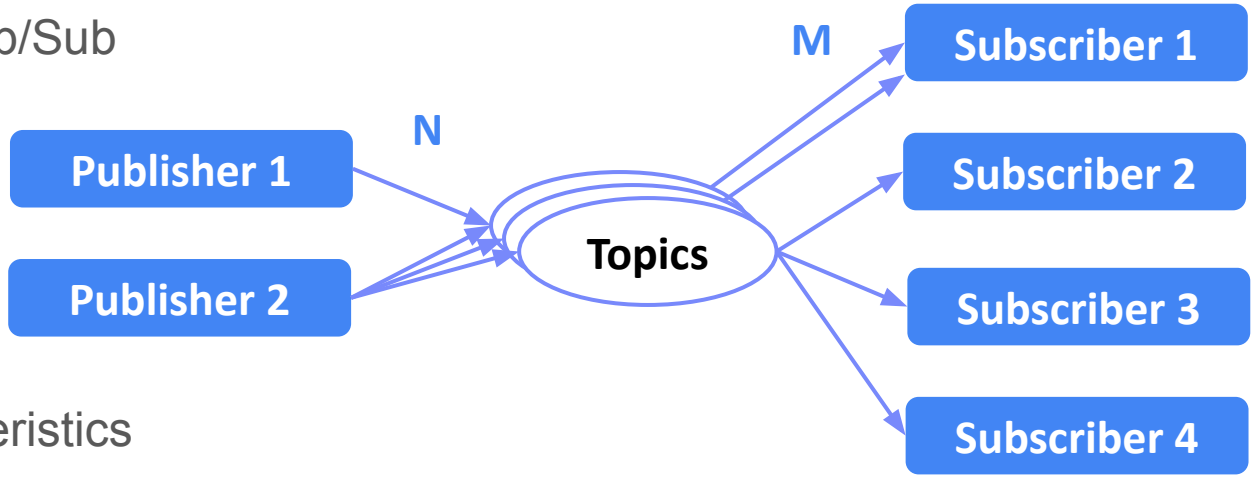


Recap: Message Delivery Guarantees

- #1 At Most Once
  - **“Send and forget”**, ensure data is never counted twice
  - Might cause data loss on failures
- #2 At Least Once
  - **“Store and forward”** or acknowledgements from receiver, replay stream from a checkpoint on failures
  - Might create incorrect state (processed multiple times)
- #3 Exactly Once
  - **“Store and forward” w/ guarantees** regarding state updates and sent msgs
  - Often via dedicated transaction mechanisms

Publish/Subscribe Architecture

- Overview Pub/Sub

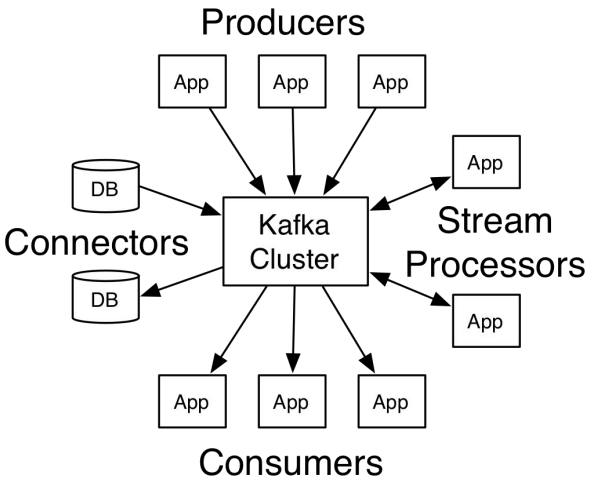


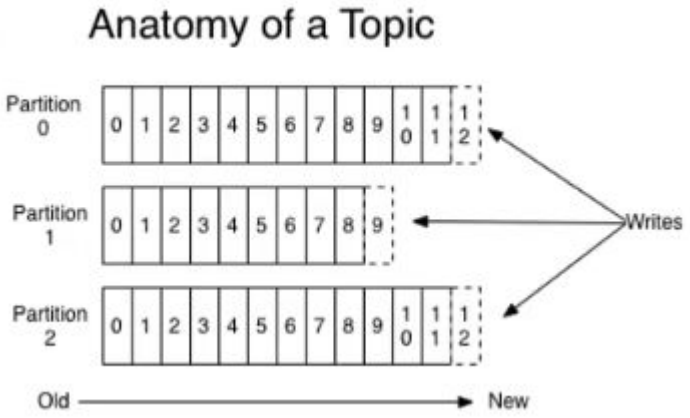
- Key Characteristics
  - Often imbalance between few publishers and many subscribers
  - **Topics:** explicit or implicit (e.g., predicates) groups of messages to publish into or subscribe from
  - Addition and deletion of subscribers rare compared to message load
  - ECA (event condition action) evaluation model
  - Often **at-least-once** guarantee

Apache Kafka

[https://kafka.apache.org/documentation]

- Overview System Architecture
  - **Publish & Subscribe** system w/ partitioned topics
  - **Storage of data streams** in distributed, fault-tolerant cluster (replicated)
  - Configurable **retention periods** (e.g., days)
  - **APIs:** producer API, consumer API, streams API, Connector API
  - **Topics:** explicit categories with user defined (semantic) partitioning

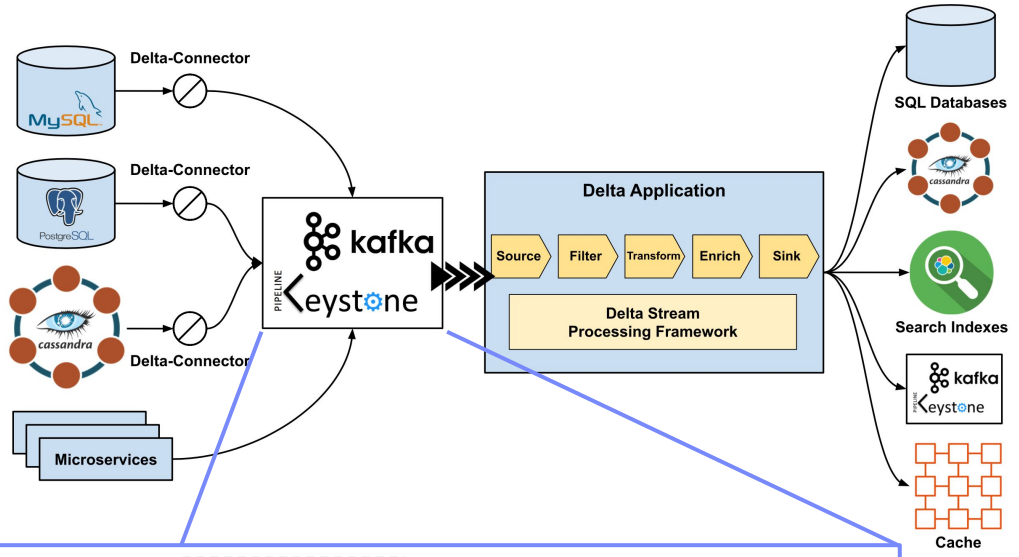


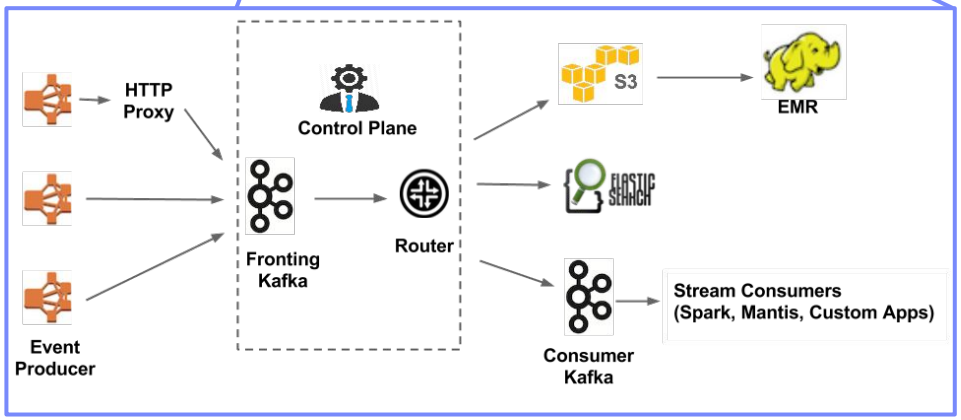


Apache Kafka, cont.

[https://medium.com/netflix-techblog/delta-a-data-synchronization-and-enrichment-platform-e82c36a79aee, Oct 15 2019]

- Netflix Delta
  - **A Data Synchronization and Enrichment Platform**
  - DSL and UDF APIs for custom filters and transformations



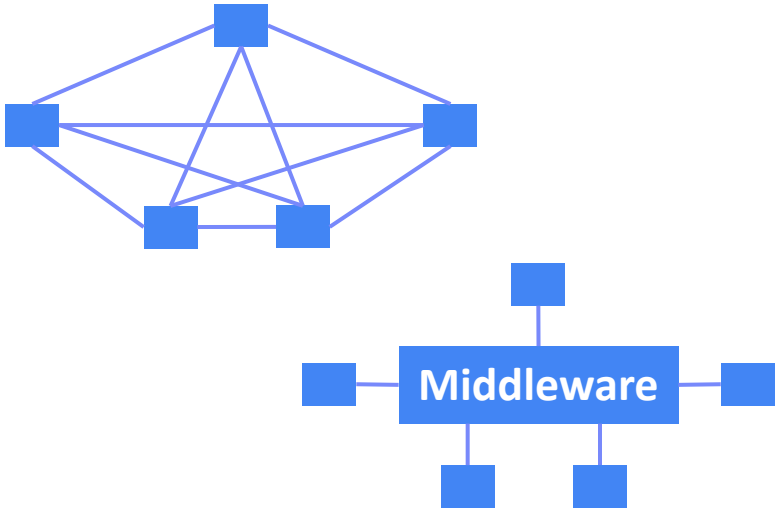


- Netflix Keystone (Kafka frontend)
  - **~500G events/day** (5M events/s peak)
  - **~1.3PB/day**

[https://medium.com/netflix-techblog/evolution-of-the-netflix-data-pipeline-da246ca36905]

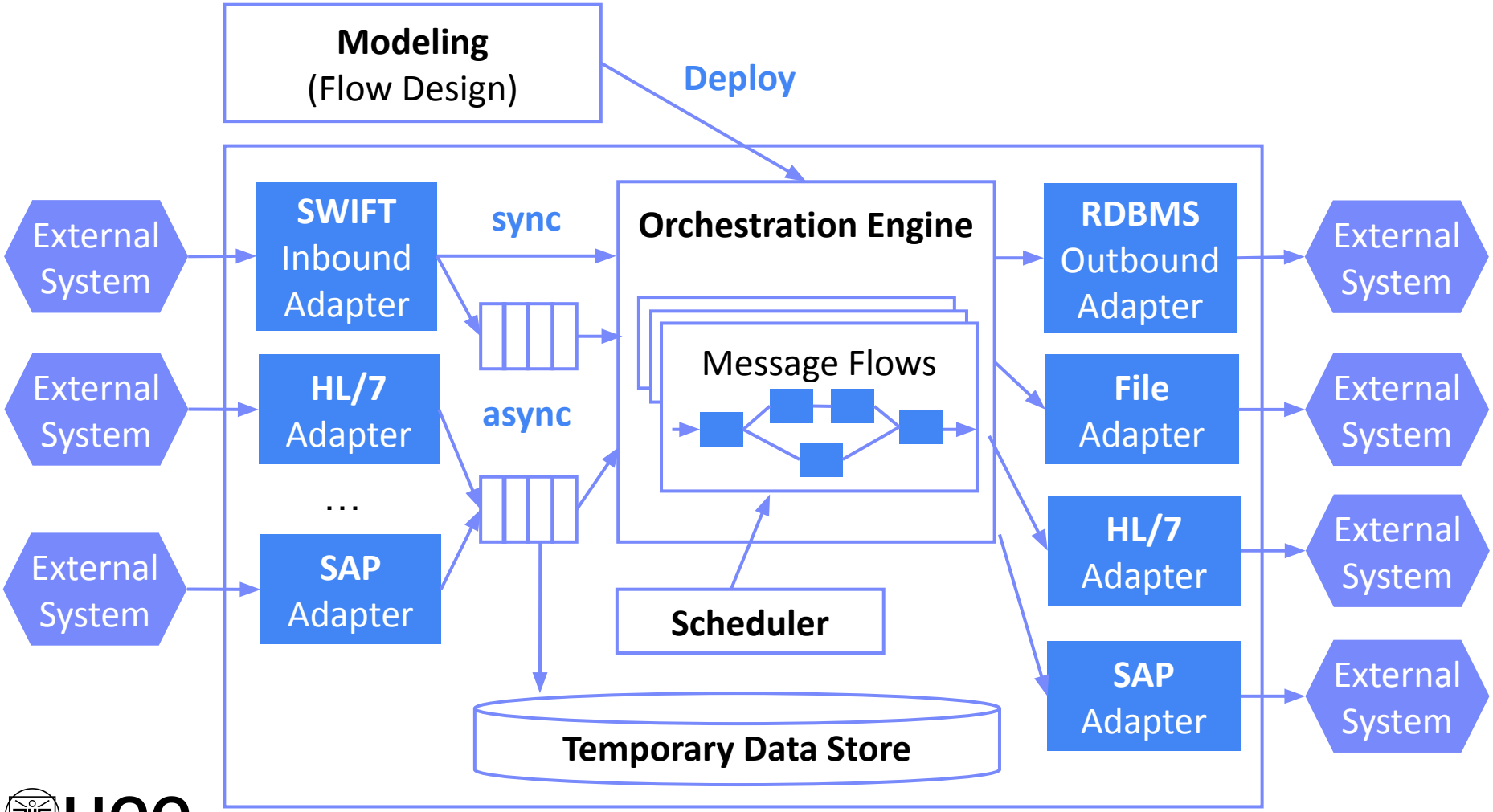
Message-oriented Integration Platforms

Overview



- Motivation
  - Integration of many applications and systems via common (information resources) IR
  - **Beware:** syntactic vs semantic data models
- Evolving Names
  - **Enterprise Application Integration** (EAI)
  - **Enterprise Service Bus** (ESB)
  - **Message Broker**
- Example Systems
  - IBM App Connect Enterprise (aka Integration Bus, aka Message Broker)
  - MS Azure Integration Services + Service Bus (aka Biztalk Server)

Common System Architecture



Common System Architecture, cont.

- #1 Synchronous Message Processing
  - **Event: client input message**
  - Client system blocks until message flow executed to output messages delivered to target systems
- #2 Asynchronous Message Processing
  - **Event: client input message from queue**
  - Client system blocks until input message stored in queue
  - Asynchronous message flow processing and output message delivery
  - Optional acknowledgement, when input message successfully processed
- #3 Scheduled Processing
  - **Event: time-based scheduled** message flows (cron jobs)
  - Periodic data replication and loading (e.g., ETL use cases)

Summary

- Distributed TX & Replication Techniques
  - Distributed commit protocols
  - Different replication techniques
- Asynchronous Messaging
  - Message queueing systems
  - Publish/subscribe systems
- Message-oriented Integration Platforms
  - System architecture and systems
- Next Lectures (Data Integration Techniques)
  - **04 Schema Matching and Mapping** [Oct 23]
  - **05 Entity Linking and Deduplication** [Oct 30]