1. Introduction
   - I. DS def
   - II. Example Data Science
   - III. IS axis
   - IV. IS evolution - 5 types
   - V. Service abstractions

2. Distributed Information System
   1. Classification of Database servers
   2. Fat Client Concepts: ESQL + CLI
   3. Fat Server Concepts: Stored Procedures
   4. Middleware
   5. Distributed object
   6. Cloud Computing

3. NoSQL
   1. NoSQL: def
   2. Types of NoSQL: 4
   3. RDF + Triple Store
   4. In-Memory DB
   5. Polystore
   6. Distributed Data Managment (CAP, Consistency, BACE, PACEL)

4. Distributed Data Infvastructure
   1. Data Managment in the Cloud (Google File System, Google Spanning
   2. MapReduce
   3. Apache Spark
   4. Data Stream Processing

5. Advanced Transaction model
   1. Semantically Rich Transaction
   2. Unified Theory (RED, PRED, SOT)

3. Multi-level transactions (Traditional, Closed Nested, Open Nested, Composite)

① Introduction

1. def **Distributed System** – multi-tier system at which at least the data managment layer or the application is physically / logically distributed across diff. nodes.

- distributed database
  - data partitioning
  - data replication (replica managment and coordination)
- distributed control
  - partitioned application services → ±
  - replicated app. services → ±

2. Example: Data Managment Aspects in DS

(Engineering → Preparation → Analytics)

- Engineering: data streams
- Preparation: data integration, cleaning
- Analytics: distributed data infrastructures

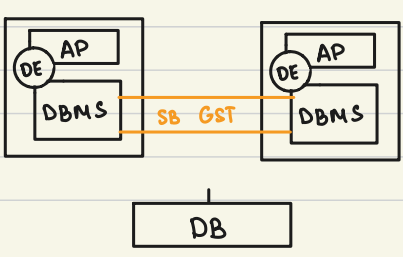
4. Evolution of Information Systems

- Infrastructure of IS's can be decomposed:

- data ⑤
  1. Relational Database System
  2. Object-Relational
  3. Distributed DS
  4. Big Data
  5. Higher-order DS
- software ④
  1. Software Systems
  2. Services
  3. Process-aware Systems
  4. Virtualization
- communication ⑤
  1. Sockets
  2. Remote Procedure Call
  3. Distributed Object Management
  4. Dynamically Deployable Code
  5. Peer-to-peer Interaction

- So evolution of IS:

① Archaic (One-tier Architecture) – C. Bachman

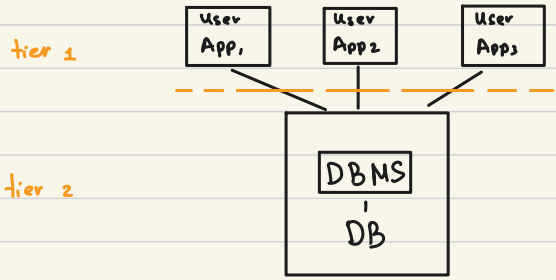


← DBMS and app. programs in the same space

- DE – memory for data exchange
- SB – system buffer of DBMS
- GST – global sync table

- Example: CICS

② Horizontally Distributed IS (Two-tier Client\Server Arch.) – Codd

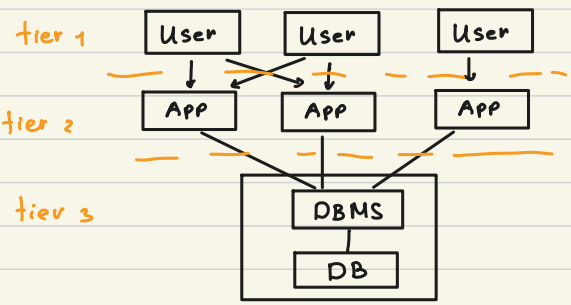


- Requirements:
  - data definition
  - data manipulation
  - transactions

⊕ data independence

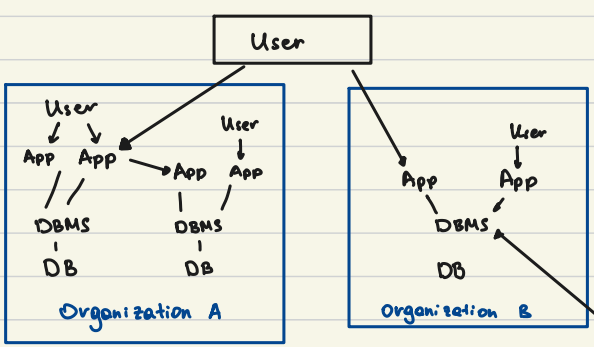
③ Three-tier (Horizontally and Vertically Distributed) – J. Gray

- decouple of app logic and user interface



- service-orientation
- no way to combine services

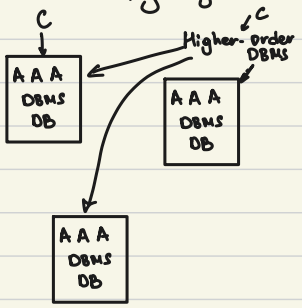
④ Distributed Information Systems (Multi-tier Client/Server Arch.) (Lindsay)



- Services are used even across system boundaries (heterogeneity)
- not just data managment, but integration and managment (coordination)

⑤ Future DIS

- Multiply layers of IS

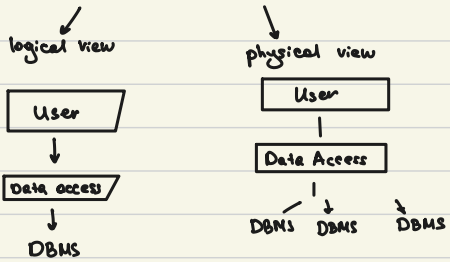


- Platform for clients concurrently accessing shared application services or shared databases.
- Higher-order DBMS:
  - Service definition
  - Service combination
  - Polyglot persistence (supports various data storage technologies)

- More broad separation of data usage from Storage

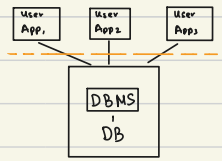
5. Service Abstraction (Control is completely delegated to a service provider)

- XaaS - everything as a service (hide everything behind a service interface)
- DaaS - data as a service



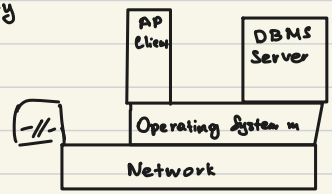
② Distributed Information System

1. From Centralized to Distributed DBMS (Two-tier architecture)



Classification of Database Servers

- Monolithic
  - one server process for many clients, server uses several threads
  - communication by shared memory



  - MS SQL Server

- Multiple Servers
  - Symmetric
    - DBMS-group of different process
    - one server process to each client
    - degree of parallelism = n
    - Oracle
    - DB2
  - Asymmetric
    - a client connected to a free server process by dispatcher
    - degree of parallelism > n
    - Oracle

2. Distribution of task

- Where application logic?

Fat Client / Thin Server

Thin Client / Fat Server

3. Fat Client Concepts

a) Embedded SQL (ESQL)

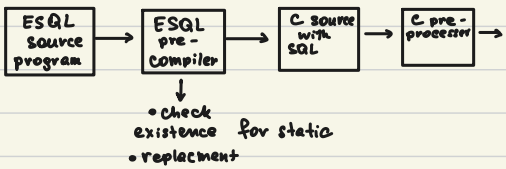
- Allows SQL statements to be integrated into a host programming language (so combination of database manipulation and general-purpose programming)
- Pre-compiler replaces SQL calls by calls of the actual runtime library of the database server.

↙ Static ESQL

- all sql statements have to be known at compile time
- all database objects have to be avaliable at compile time

↘

- SQL statement are generated at runtime
- the necessary database objects need not to exist



3) SQL Call-Level-Interface (CLI)

- Standardized programming interface allows apps to interact with database managment system dinymically at runtime.
- Comprises:
  1. connection
  2. preparation and execution SQL statements
  3. binding of parameters
  4. take results
- CLI-Dialects

1. ODBC (Microsoft)
2. Proprietary CLIs of DB manufactuves ("Native API") OCI
3. Object CLI - extention of CLI to object interfaces - JDBC

4. Thin Client / Fat Server

- Application Logic inside a db
- Stored Procedures (TP-Lite)
  - 4Gl
  - Turing Complete
  - Storage and execution inside database
- Triggers
- Stored procedures → External / Internal

- ± of Thin / Fat Server / Client

2. Three-Tier / Multi-Tier Systems

- Application logic now third tier, it is not part of client or server.

=> Three-Tier

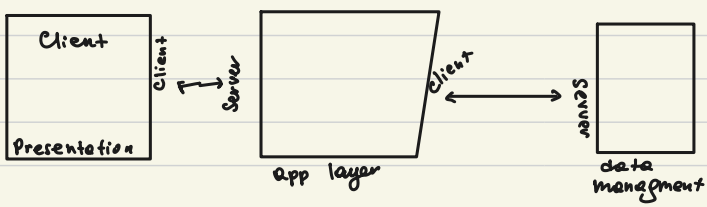
- Separation of app logic to several components that mutually interact =>

=> multi-tier architectures

For this we need:

- to combine all this components in case of physical distribution and heterogeneity.

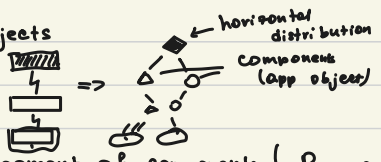
Middleware



- Application development is completely independent of the DBMS manufacture
- Middleware supports the development and execution of applications without directly providing business-specific functionality (offers only essential for separation)
- Communication middleware
  - ↳ RPC (Remote Procedure Call) – communication between different processes

- Middleware Services (on top of basic communication)
- Middleware Frameworks

- Distributed Objects



- component base system

– Development of components („Programming-in-the-Small")

– Development with components („Programming-in-the-Large)

3 Cloud Computing

XaaS
- XaaS
- SaaS
- DaaS
- PaaS

- Pay-as-you-go + Virtualization - One most important properties
- Elasticity
- AWS + Azure + Google Cloud Platform
- Hybrid Clouds

③ NoSQL

1 NoSQL (Not only SQL)

- No schema + inherently distributed + highly scalable

2. Types of NOSQL Systems

a) Key-value Stores

- <key, Blob> [It's app task to interpret the content of BlOB]
- Operations:

I. Get II. Put III. Delete

- Suitable
  - Not
    - search by data
    - referential integrity
  - yes
    - web app (key: user_id)
    - real-time recommendations (user_id)

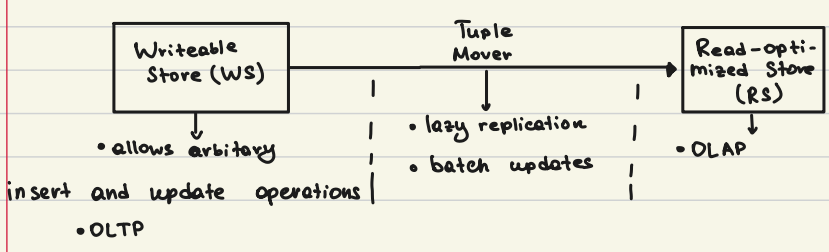
- Examples: Voldemort, DynamoDB

- def OLTP - online transaction processing - captures, stores and processes data from transactions in real time

def OLAP - online analytical processing - uses complex queries to analyze aggregated data from OLTP

b) Column Stores

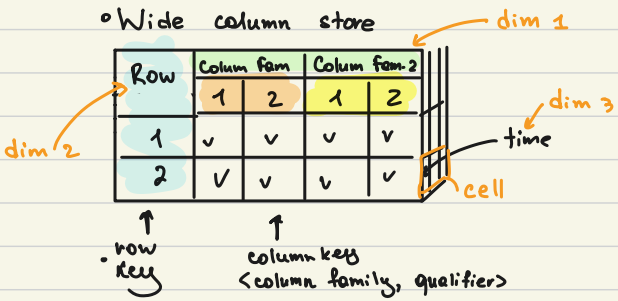
- stores data in columns together
- for OLAP, not OLTP! • Data Warehouses
- C-Store example
  - Uses a two-store approach



- Logical and physical data model:
  - Relational model as its logical data model
  - Data stored in projections as physical storage layer

- RS organization
  - For each projection Sort key
    - each projection partitioned into segments with SID (Segment Identifier)
    - The attributes of each segments have Storage Key (SK)
    - Link to other attribute with the same tuple done via Join indexes

- Example: Google BigTable (wide column store)
  - designed to a very large size (very high read\write rate, thousands of servers, diff. data size)
  - Wide column store



Examples: BigTable, Cassandra, HBase

B) Document Database
- <record, unique_key>
- record can be (they can be nested)
  - I. JSON: <key, list of values>
  - II. XML: elements with attributes
- Can be retrieved by key or by content (using API or query language)
- Applications: Text docs, Metadata

Examples: MongoDB

r) Graph Database

- Graphs are structure to manage networked information

Concepts about graph

I.
- Direct/Undirect
- Multigraph
- Weighted graph

II. Graph Algos
a) Cycle detection
b) Eulerian path/cycle(edge)
c) Hamilton path/cycle(node)
d) Spanning Tree
e) DFS, BFS
f) Shortest Path

• Graph Data Structure

I Edge list

Graph: set_of_nodes V + set of edges

- set of sets for undirected graphs
- set of tuples for directed graphs
- multiset of sets

II. Adjacency Matrix

$$a_{i,k} = \begin{cases} 0 & \text{no edge} \\ 1 & \text{edge} \\ n & \text{multiple edges} \end{cases}$$

III. Incident Matrix

$$b_{i,k} = \begin{cases} 0 & \text{if node and edge not connected} \\ 1 & \text{else} \end{cases}$$

vertic.	edges

IV. Adjacency List

$v_1 \rightarrow v_2 \rightarrow v_3$

v_2

$\rightarrow v_2 \rightarrow v_1 \rightarrow v_1$

V. Incidence List

$V_1 \rightarrow e_1 : v_2 \rightarrow e_3 : v_3 \rightarrow e_4 : v_3$

...

- **Property Graph Model**
  - Directed Multigraphs
  - V and E also have further info
  - Nodes and edges are typed with corresponding set of attributes (name:value => properties)

def Property graph: $P = (V, E, L_V, L_E, ID)$

(L_E: set of edge labels; L_V: set of nodes labels)

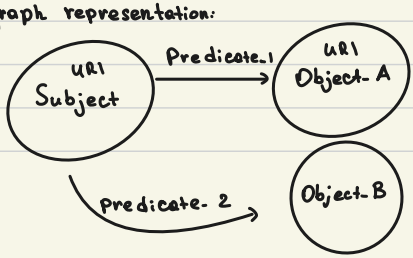
- Storing:
  - one table for nodes + one table per node type (for attributes)
  - one for edges + one table for each edge type
- Advanced graph models
  - hypergraphs with hyperedges
  - Nested graph with hypernodes
- **Example: NeoJ**
  - Property graph model
  - edges = relationships
  - ACID
  - Apache Lucene indexing
  - Cypher language:
    - declarative
    - nodes: ( ), relationships -->, <--, --,
    - match, where, return

Application: Bioinf, Social networks

Examples: NeoJ

3. RDF (Resource Description Framework)

- properties of resources (URI)
- data model for metadata; self-descriptive data
- RDF graph representation:



- Triple Store DB
  - DB only supports RDF triples
  - named triples? → Quad stores

4. In-memory DataBase
   - main memory is the primary location
   - ACI, no D
   - can be applied to NOSQL\Relational system
   - To survive system crashes:
     - snapshots
     - logging
     - replication

Examples: Apache Derby, SAP Hana

5. Polystores
   - Combine polyglot persistence and multistores
   - Different storage technologies
   - Data replication
   - Decision per query where to route a request
   - Logical database consisting of several physical databases

Examples: Polypheny

6. Distributed Data Managment in the Cloud
   - Consumer Perspective
     - Service level agreements
     - Elasticity
   - Provider's Perspective
     1. Capasity utilization
     2. Multi-tenancy
     3. Quality of Service: Replication + Performance
     4. Managment: read operations with diff. semantics + long-term preservation and archiving

Different level of consistency

- Strong consistency (eager replication)
  - limited availability
- Weak consistency (lazy replication)
  - full availability

Requirements for Cloud Data Management

a) 1. Tolerance to network partitions (network operates despite network partitions)
2. Availability (every request receives a response)
3. Consistency (every read receives the most recent write)

5) CAP Theorem

- Trade-off between availability and consistency

I. Data Replication →
- W+R > N (and $W > \frac{N}{2}$) → strong consistency
- R+W = N => weak consistency or eventual consistency
  ⇓ high availability means N > 2 (N=3)
  + different combinations (fault-tolerance, high read loads etc.)

B) Consistency

Strong:

1 Sequential Consistency (Serial execution, CPSR)
2. Casual Consistency (Writes causally related in the same order)

Weak:

1. Monotonic Read
2. Monotonic Write
3 Read-your-Write
4. Writes-follow-reads
5. Session Consistency

- Eventual Consistency (form of weak consistency)
  - Guarantee
  - Might return inconsistent data

Example: Amazon's Read Replicas

- ASID → BASE (Basically Available, Soft state, Eventual Consistency) „D (will be expired)
- PACELC (non-failure case, CAP is for failures)
  CAP Else Latency vs Consistency

④ Distributed Data Infrastructures

a) Data managment in the Cloud

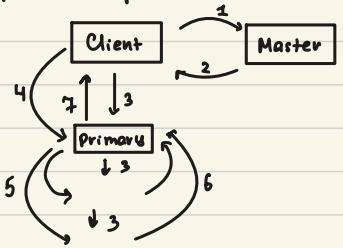
def Big data:

- Velocity
- Volume
- Variety

- Cloud Infrastructure
  - Distributed file systems
    - large number of servers
      - robust against hardware failures

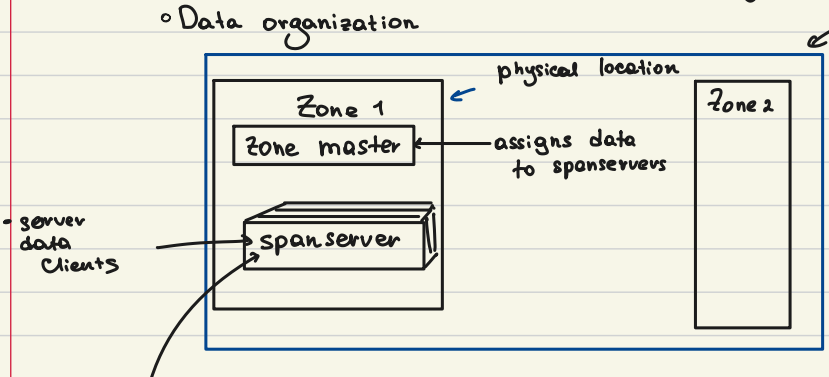
- Cloud File Systems
  - Workload
    - Large stream reads
    - Small random reads
    - special write: mutation append data to files

- Example: Google File System (GFS)
  - Cloud Data Managment
  - + 2 operations (Snapshot + Record append)
    - Snapshot: • System operation
    - Record append: • concurrent, but with consistency • record append
  - Architecture
    - Single master server + several Chunk servers (with Linux file System) store locally as Linux files
    - files are subdivided into chunks of fixed size 64 mb
    - triplication (heartbeat)
    - master keeps all metadata
  - Data Access
  - Consistency: Lazy replication + append-only
  - Update Operations



- Database in the Cloud: Google Spanner (NewSQL database)
  1. ACID
  2. SQL – based
  3. Scalable
  4. Multi-version
  5. Globally distributed
  6. CAP restricted (but due to replication very aval.)

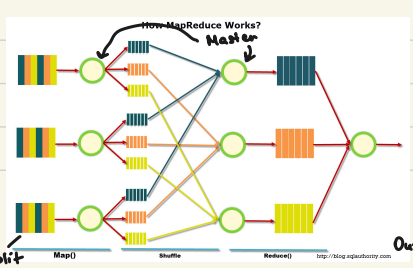
- Data organization



- each manages 100-1000 tablets: (key, timestamp)
  - single paxos state machine
- set of tablets that refer to the same data is Paxos group. (with Paxos Leader)
- if more than one group → 2PC
- Spanner transactions:
  - Read-write
  - Read-only transactions
  - Snapshot reads
- Synchronization based timestamp (reflect serialization order)
  - Global clock

6) Big Data Processing

I. MapReduce (automated parallelization on a large cluster



- Requirements
  - chunks
  - scheduling partitions across a set of machines
  - handling failures
  - handling communication

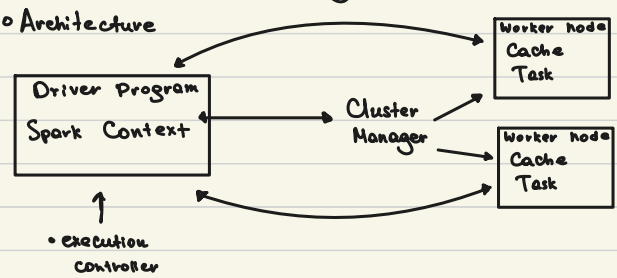
- Inverted index
- Web link graph

Apache Spark

- Generalize MapReduce
- Supports: Iterative algos + interactive data access
- RDD - Resilient Distributed Datasets
  - distributed memory abstraction (among nodes of a cluster)
  - immutable collection of objects
  - partitioned collection of data
- RDD operations:
  1. Transformations (creat a new dataset from an existing one
  - lazy, lineage of transformations to appy until (collect, count, save)
  - |
  2. Actions (Return a value or export data)
- Persistence and Partitioning
  - Cache or persist a dataset across operations

It can Control partitioning and optimize data placement

- Architecture



- Extentions: Spark SQL, Spark Streaming, MLib, GraphX

B) Data Stream Processing

- Sensors → ↓ → Vizualization
- Stream operators: Sliding Windows
- Stream Applications: Stream operator

Databases	Data Streams
Data: persistent	Queries: persistent (continuous)
Queries: transient	Data: transient (float through the system)

- Architecture
  1) Lambda (Batch queries + real-time), Serving layer from Batch, Master dataset from Batch
  2) Kappa (Real-time), Master Dataset from Online

- Spark Streaming (batch mode, adds data stream on top Apach Spark)
  - Execution modes (Complete, Append, Update)
- Apache Storm (one-data-at-a-time, newly data stream elements are processed immediately)
  - Consists of types of element:
    - Spouts (data source)
    - Bolts (processing elements)
- Apache Flink (batch + one-date-at-a-time)
  - fault-tolerance based on distributed checkpoints

(5) Advanced Transaction Model

- S2PL and 2PC are needed for correctly supporting distributed transactions

a) Semantically Rich Transactions

I.
- from Read\Write to Semantically Rich Operations
- atomic ⟶ composite
- single-level scheduling → multi-level scheduling

II. Commutativity
- for rich operations interface are known, but not the implementation.
- so the source of info: → invocation parametrs, → return values

def Commutative Operations:

p and q commute, if for all seq of operations F^1 and F^2 holds: the return values of $<F^1 \circ p \circ q \circ F^2>$ are the same as $<F^1 \circ q \circ p \circ F^2>$, $\circ$ - seq. execution of operations.

def Effect-free operation

$F^1 \circ o \circ F^2$ the same as $F^1 \circ F^2$

def Inverse Operations

$F^1 \circ (o' \circ o'') \circ F^2$

$o' \circ o^{-1}$

def Perfect Commutativity

- p and q commute, then p^{-1}, q, p, q^{-1}, p^{-1}, q^{-1} commute
- p and q are in conflict, then ... are conflict
- exception

Examples of operations: Incr, Res, Decr, etc.

- Escrow Locking
  - test whether decr (incr) can be executed; idea: if there is sufficient supply — two decr commute
  - v in [m,n]
    - incr(v,a): $v \leq (max - a) \Rightarrow [m, n+a]$
    - decr(v,a): $v \geq (min + a) \Rightarrow [m-a, n]$

B) Unified Theory of Concurrency Control and Recovery

- Atomicity + Isolation now one criterion
- Steps:

I. Expansion: define what to be undone for each transaction
- for each action define inverse action in reverse order
- all active transactions are aborted jointly
- we get expanded schedule

II. Reduction: check if these can be undone
- Reducibility of a schedule: creating a serial schedule by re-ordering actions that are not in conflict

def Reducible Schedule (RED)
- $\tilde{S}$-RED, if $\tilde{S}$ can be transformed into S_{ser}
  1. Commutativity rule
  2. Undo rule

Limitation: not suitable for dynamic scheduling, because sometimes we don't know from prefix.

def Prefix Reducibility (PRED)
- A S is PRED if S and each prefix S is reducible
- Guarantees: correct concurrent execution and correct failure handling
- (CPSR & RC) ⊇ PRED ⊇ (CPSR & ST) ⊇ RG ← locks on all until C or A
- RED ⊇ PRED

def SOT (Serializability with Ordered Termination)

A non-expanded S is SOT if

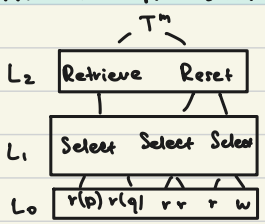
1. S is serializable (CPSR)
2. S is recoverable (RC)

and if for all conflict $w^i(x) < w^k(x)$ where T^i and T^k in direct conflict

3. if $C^k \in S$, then $C^i < C^k$
4. if $A^i \in S$, then $A^k < A^i$ (opposite order or joint abort)

- In read/write model SOT ≡ PRED

r) Multi-level Transactions



I. Traditional (Single-Level)

Input: set of independent transactions

Scheduler (Commutativity Matrix)

Output: serializable schedule

II. Closed Nested Transactions

-

- Properties: - very limited degree of parallelism
  - structure on global transactions
  - save point
  - global atomicity

III. Open Nested Transactions:

- Allow sub-transactions to commit on the spot
- Recovery more complex

- Move assumptions:

strong order → a) Conflict pairs (at each level) have to be ordered, one finished — another started (order sub-transactions)

b) This order should respected in all sub-levels (sub-transactions) of the conflict operations.

B) Extended transaction model:

- Inter-Transaction Order →
- order < on operations should be passed down to sub-transactions.

2) Correctness

a) Multi-level schedule in open nested is correct if:

- each S is conflic-preserving serializable (correct)
- serialization order of each respect

IV. Composite Transactions

- sub-transactions which implement conflicting operations can be executed in parallel (weak order)
- Model extention:
  - a S estabishes both weak and strong order

- Correctness:
  - each S is conflic-preserving serializable (correct)
  - serialization order of each S respect the strong inter-transaction and weak inter-transaction order beetween transactions at this level

⑥ Middleware

1. Middleware is a layer of software that sits beetween the OS / network-program or clients and servers.

2. MOM - enables communication between distributed applications through message passing.

Features

- Asynchronous Procedure (data + specification)
- Queued Transactions (temp. storing msg)

3. Variants of MOM:

1. Type of connection
   1. Point-to-point
   2. Pub/Sub (example Java Msg Serv.)
      ↳ 1. Topic-based 2. Type-based (ex. event)
      3. Content-based
2. ↳ Variants of Interactions:
   - Peer-to-peer
   - Hub-and-Spoke
3. ↳ Pull and Push
4. ↳ filtering
   Broker → Recipients
5. ↳ Persistence of msg
   ↳ Transaction support
- Guranteed Execution
  - Insert into Queue
  - Dequeue Request
  - Deque Response.

2. Else:
   - Load Balancing
   - Priority