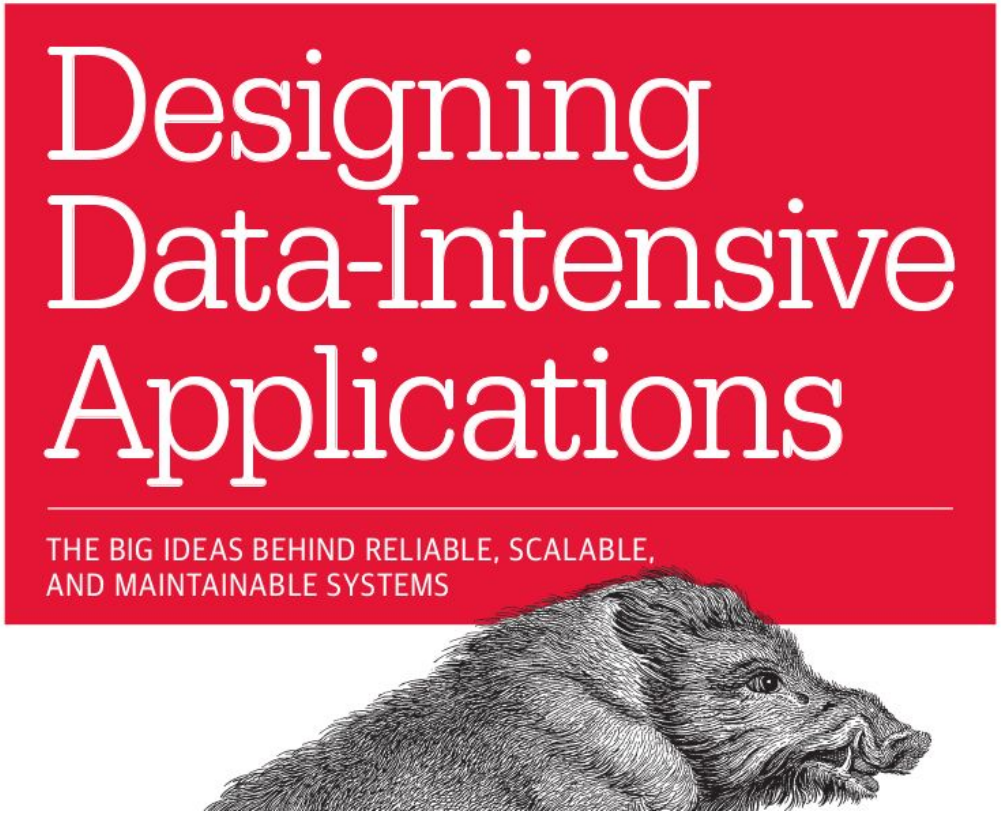


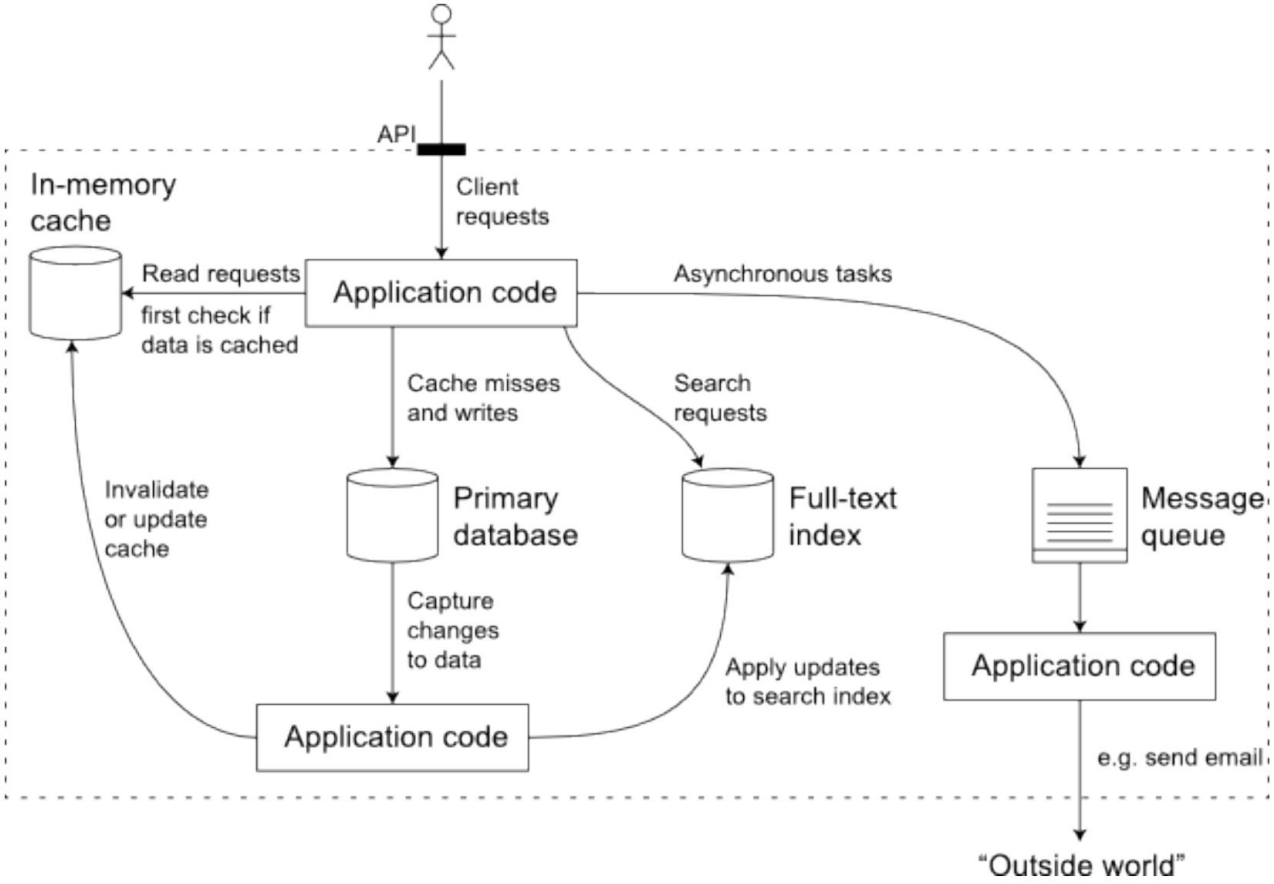
CHAPTER 1

RELIABLE
SCALABLE
MAINTAINABLE

Is yours a Data-Intensive use case?

An application is data intensive if the **amount of data that it generates**/uses increases quickly or if the **complexity of data** that it generates/uses increases quickly or if the **speed of change in data** increases quickly.

Components of data intensive app



Credit: Martin Kleppmann: Source https://www.semanticscholar.org/paper/Designing-Data-Intensive-Applications%3A-The-Big-and-Kleppmann/24f14e3b30012c2bc7e3abbdb16e2b3365d6f920

Main components in a data-intensive app:

1. **Databases** for source of truth for any consumer. E.g. Oracle/MySQL
2. **Cache** for temporarily storing an expensive operation to speed up reads. E.g. Memcache
3. **Full-text Index** for quickly searching data by keyword or filter.e.g. Apache Lucene
4. **Message queues** for messaging passing between process. E.g. Apache Kafka
5. **Stream processing**. E.g. Apache Samza, Apache Spark
6. **Batch processing** for crunching large amount of data. E.g. Apache Spark/Hadoop
7. **Application code** which acts as the connective tissue between the above components.

Role of an application developer is to design the data system for **reliability**, **scalability** and **maintainability**.

Reliability

- Fault tolerance
- No unauthorized access
- Chaos testing
- Full machine failures
- Bugs - Automating tests
- Staging/testing environment
- Quickly roll-back

Scalability

- Handle higher traffic volume
- Traffic load with peak #of reads,writes and simultaneous users
- Capacity planning
- Response time vs throughput.
- End user response time
- 90th,95th Percentile SLO/A service level objectives/agreements.
- scaling up (more powerful machine)
- scaling out (distributed many smaller machines).

Maintainability

- Add new people to work
- productivity
- Operable: Configurable and testable
- Simple: easy to understand and ramp up
- Evolveable: easy to change

There is no one magic solution that works for all data-intensive application use cases. **Your solution will be custom** to your needs, as long as you have a good idea of the above three pillars of Reliability, Scalability and Maintainability you will build the right system.

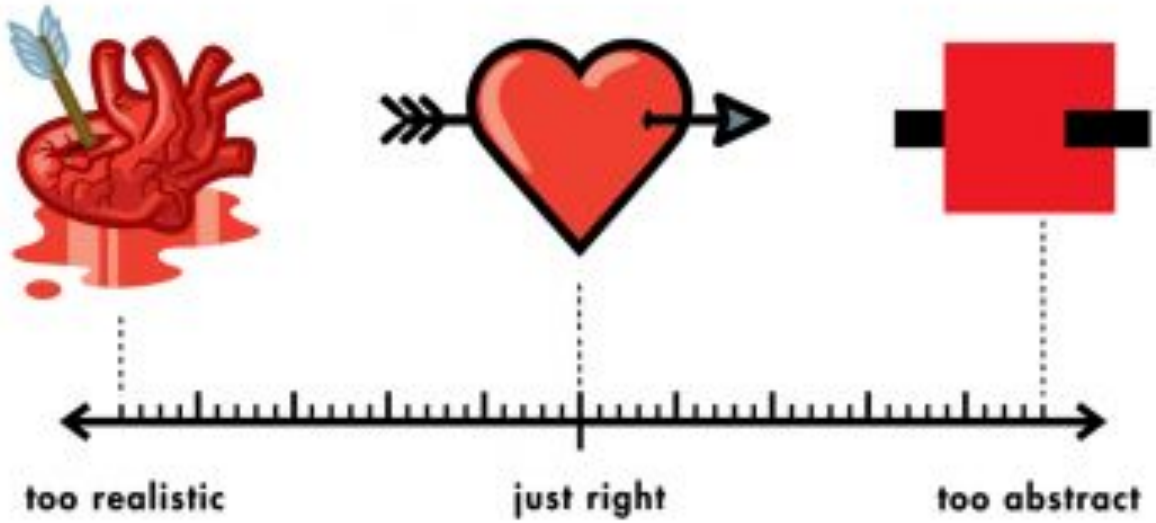
Designing Data-Intensive Applications

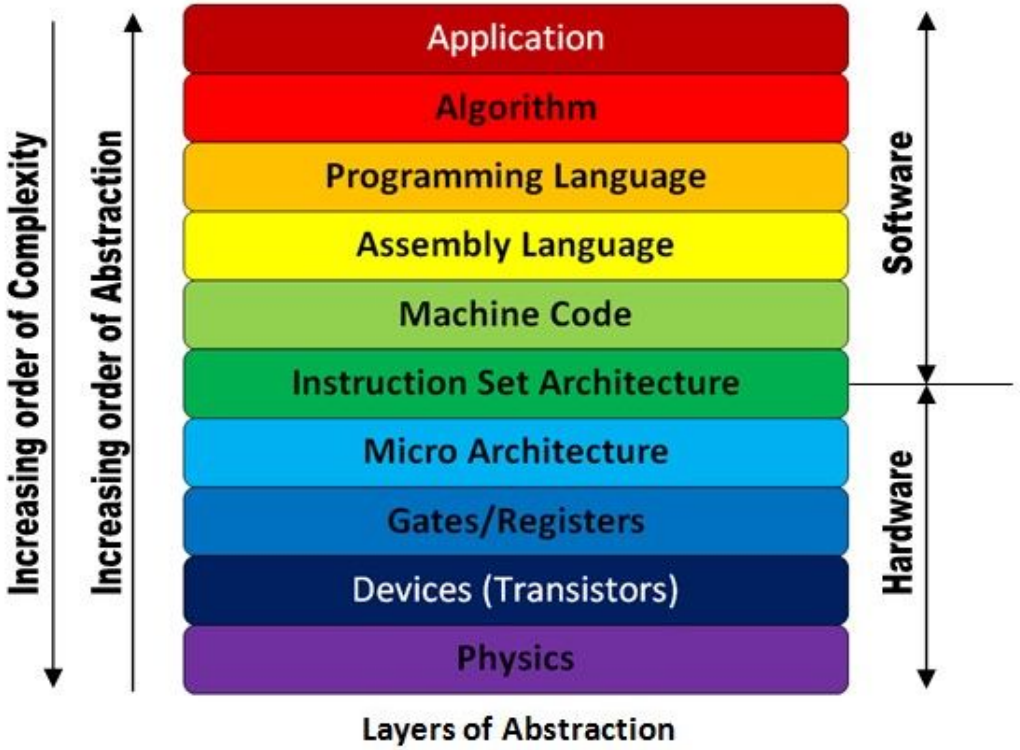
THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS

CHAPTER 2

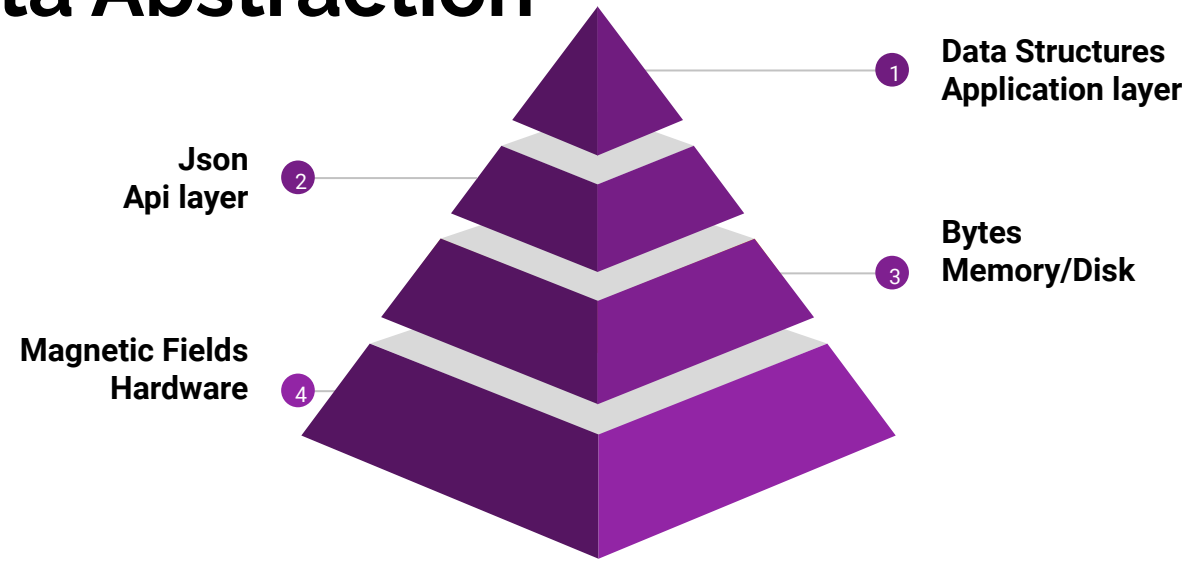
DATA MODELS & QUERY LANGUAGES

THE ABSTRACT-O-METER




Application
Algorithm
Programming Language
Assembly Language
Machine Code
Instruction Set Architecture
Micro Architecture
Gates/Registers
Devices (Transistors)
Physics
Increasing order of Complexity
Increasing order of Abstraction
Software
Hardware
Layers of Abstraction

Data Abstraction

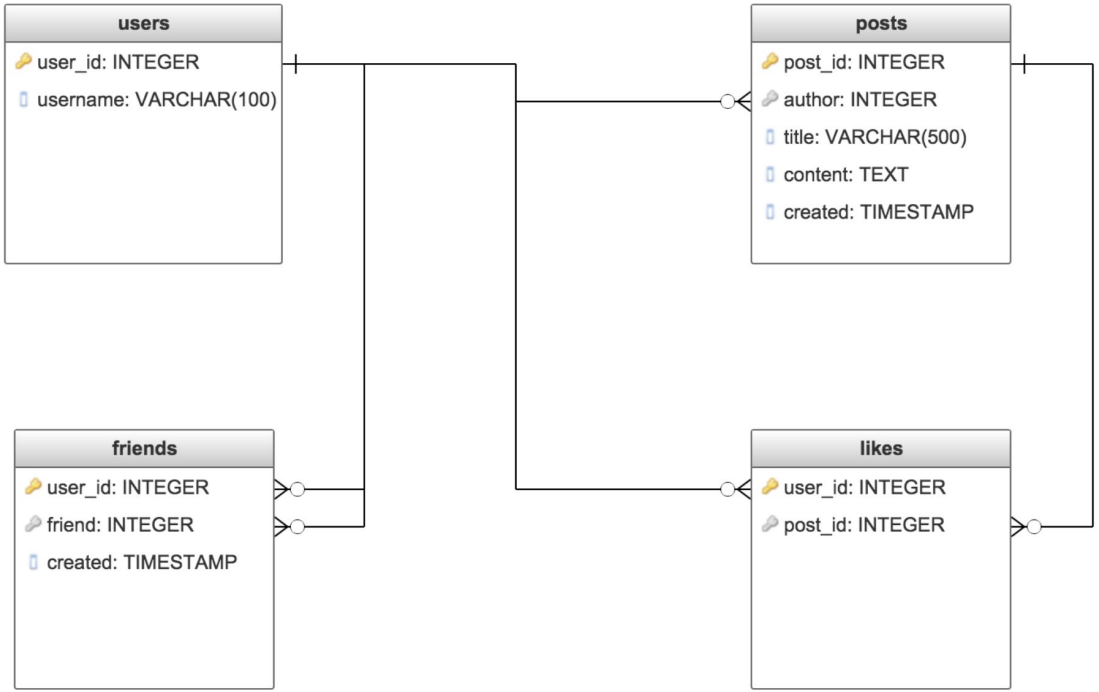


Model a social network
Users -> Friends -> Posts -> Likes

Relational model

Relational

In a typical Relational Database, this will likely be modeled using four different tables - `users`, `posts`, `friends` and `likes`. These might look something like this:



Relational query

```
SELECT friends_of_likers.*
FROM posts
JOIN likes ON (posts.post_id = likes.post_id)
JOIN users likers ON (likers.user_id = likes.user_id)
JOIN friends ON (likers.user_id = friends.user_id)
JOIN users friends_of_likers ON (friends_of_likers.user_id = friends.friend)
WHERE posts.author = :me
ORDER BY friends_of_likers.username ASC
```

Document based model

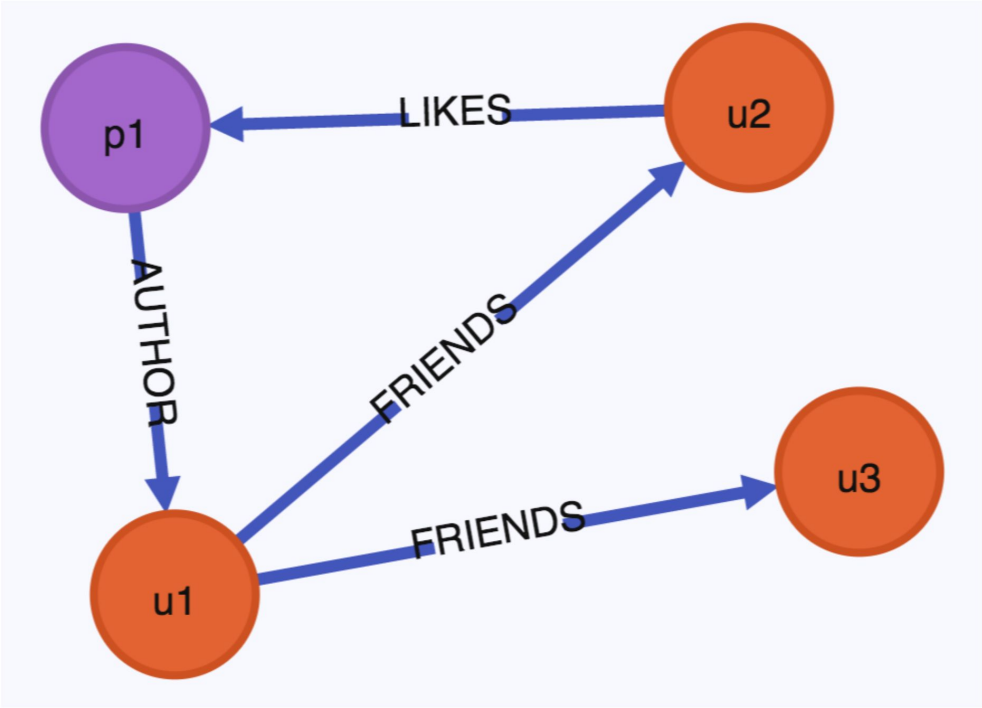
Users

```
{
  "user_id": "u1",
  "username": "grahamcox",
  "friends": {
    "u2": "2017-04-25T06:41:11Z",
    "u3": "2017-04-25T06:41:11Z"
  }
}
```

Posts

```
{
  "post_id": "p1",
  "author": "u1",
  "title": "My first post",
  "content": "This is my first post",
  "created": "2017-04-25T06:41:11Z",
  "likes": [
    "u2"
  ]
}
```

Graph model




```
MATCH (:User {id:{author}}) <-[:AUTHOR]- (:Post) <-[:LIKES]- (:User) <-[:FRIENDS]- (u:
```

Data Storage and **Data Retrieval** are the two main things to consider. Same query can be either written in 4 lines or with 30 lines based on your data model!

3 Main categories of databases

01	**Relational Database**	• Optimized for Transaction and batch processing (read throughput), joins.. • Data organized as table/relations • Object Relational Mapping needed. • Oracle, MySQL, PostgreSQL..
02	**Document Database**	• NoSQL - Not only SQL. • Flexible schemas, better performance due to locality/High write throughput. • Mainly free and open source. (Espresso,CouchDB, MongoDB..)
03	**Graph Database**	• Best suited for highly interconnected data - many to many relationships. • Social graph, Web graph (Neo4j AnzoGraph, SPARQL, Cypher)

Document databases target use cases where data comes in self-constrained documents and relationships between one document and another are rare.

Graph databases go in the opposite direction, targeting use cases where anything is potentially related to everything.

Query languages

Imperative - e.g. SQL - tell exactly how
to perform certain operation in a certain order.

SELECT * FROM animals WHERE family = ‘Sharks’

Declarative - e.g. CSS - just specify the pattern

Li.selected > p { background-color: blue; }

?map reduce?


mongoDB®

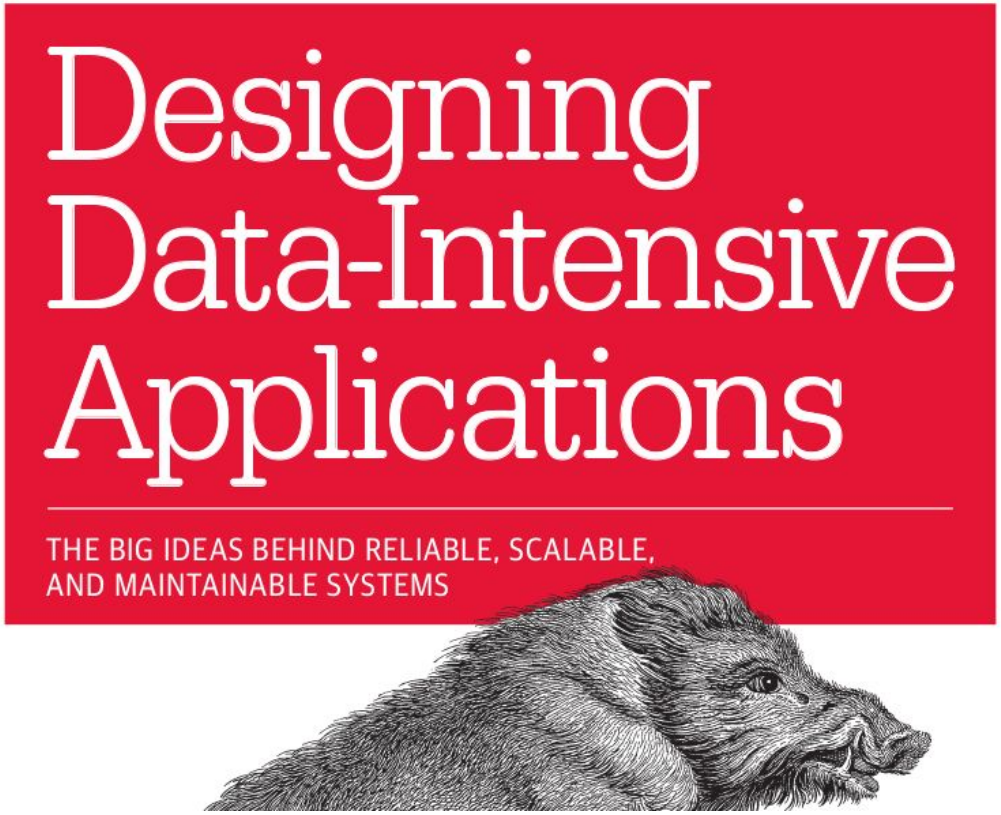

MySQL®


Couchbase


PostgreSQL


neo4j

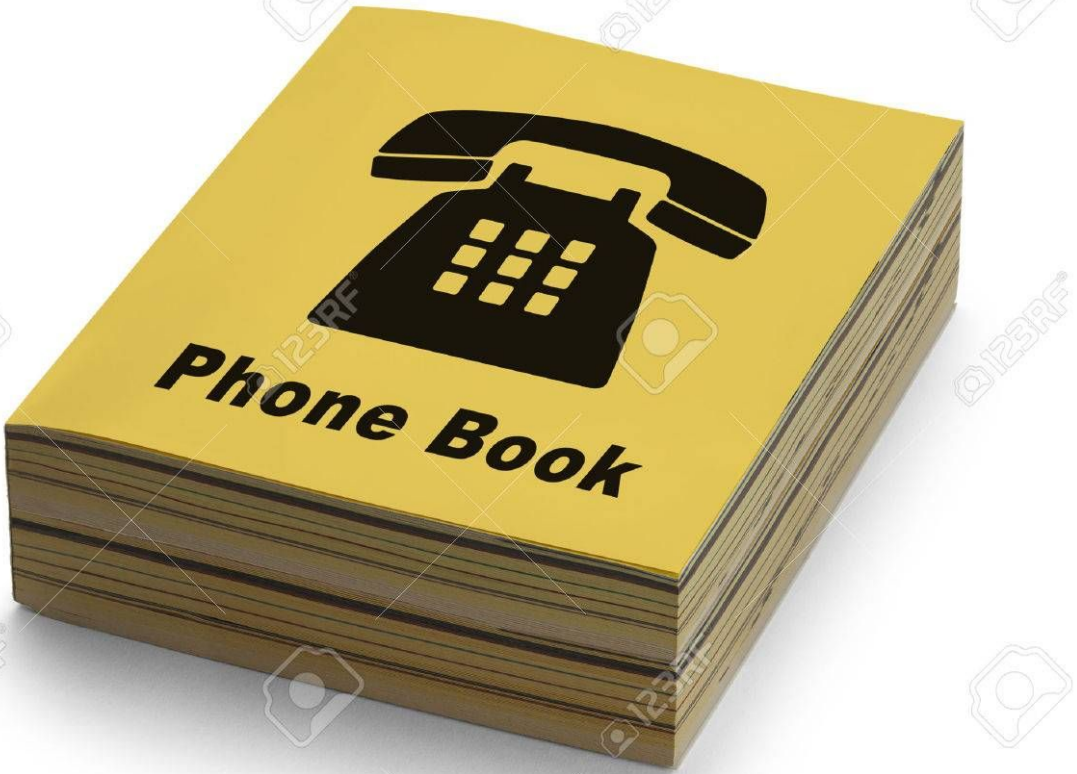

ArangoDB



CHAPTER 3

STORAGE & RETRIEVAL

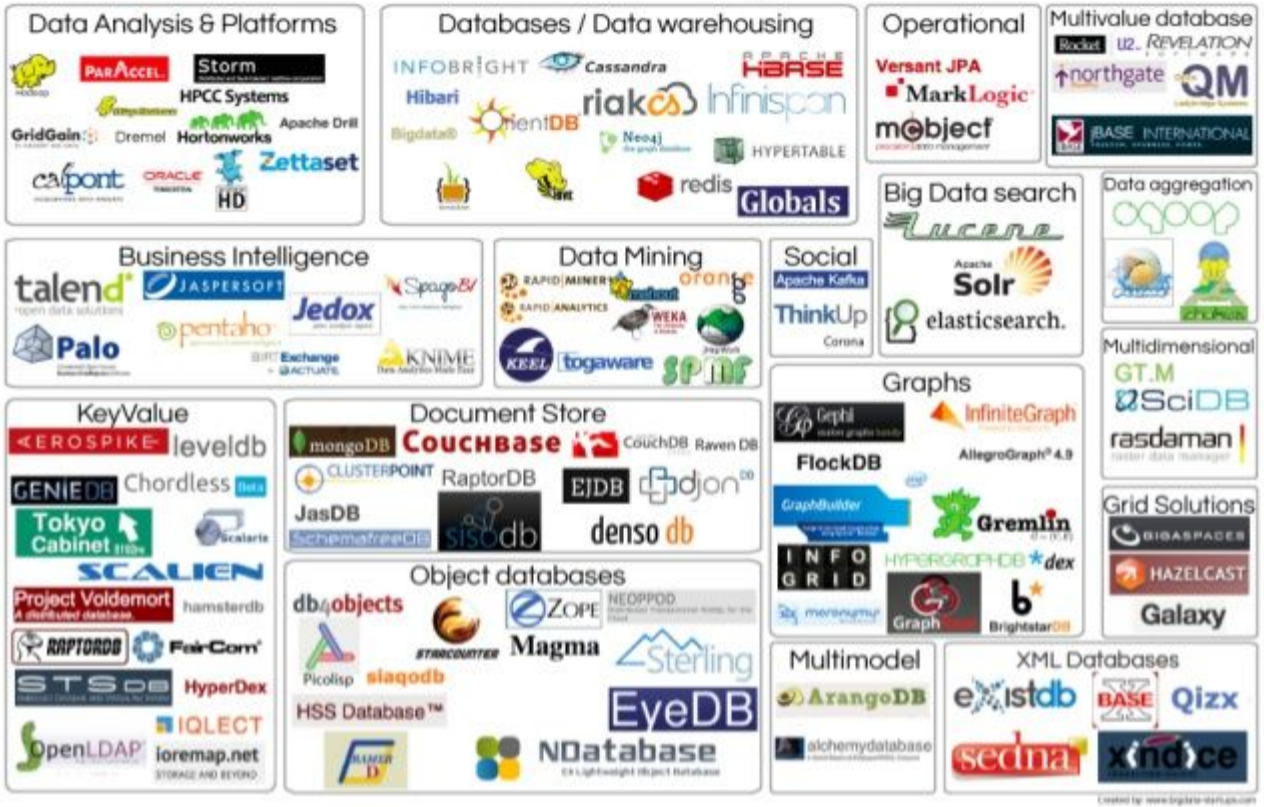
Storage & Retrieval



Which database to use ??

- Every storage engine is optimized for different use cases. Select the right storage engine for your use case.
- As an application developer we need to have a rough idea on what the storage engine is doing under the hood so that we can select the right one.
- Tuning and optimizing pointers.

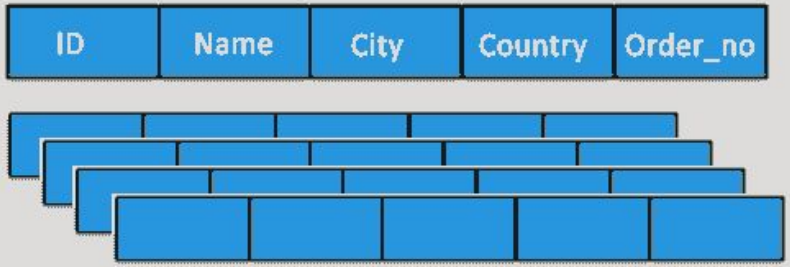
BigData Tools: noSQL Movement



Which database to use ??

- There are two broad categories of databases - OLTP and OLAP each with different read pattern, write patterns, user using it, data size etc.
  - OLTP - Online Transaction processing database are optimized for latency.
    - E.g. MySQL, Oracle.
  - OLAP - Online Analytical processing databases are optimized data crunching! Data Warehousing - Star/Snowflake schema, Column oriented! Column compression, Data Cubes, optimized for reads/queries. Materialized views. Lack of flexibility.
    - E.g. Hbase, Hive, Spark,

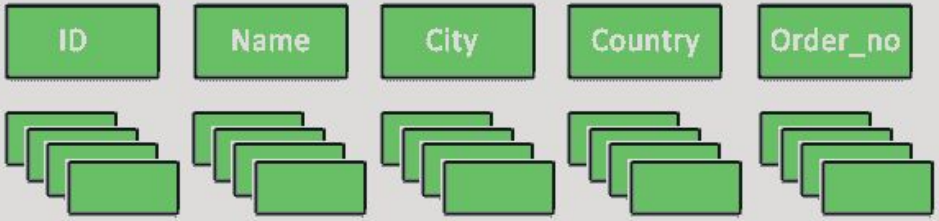
row-store



+ easy to add/modify a record

- might read in unnecessary data

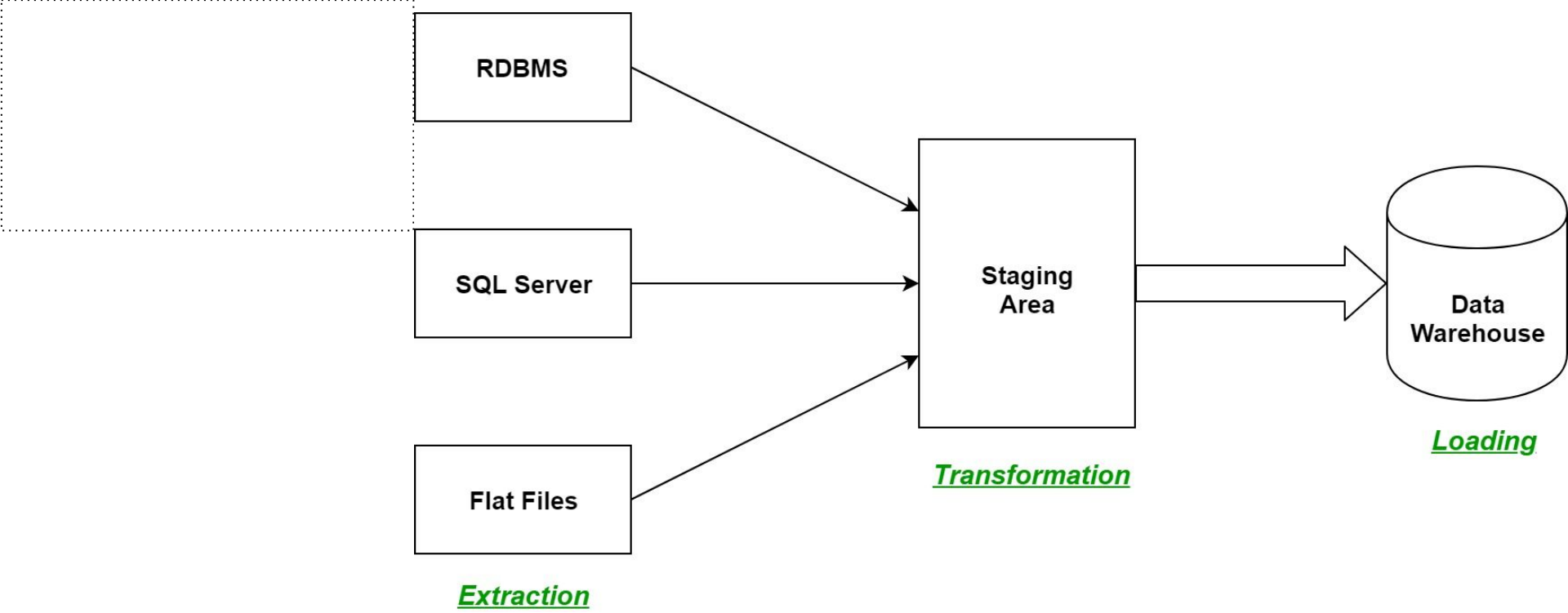
column-store

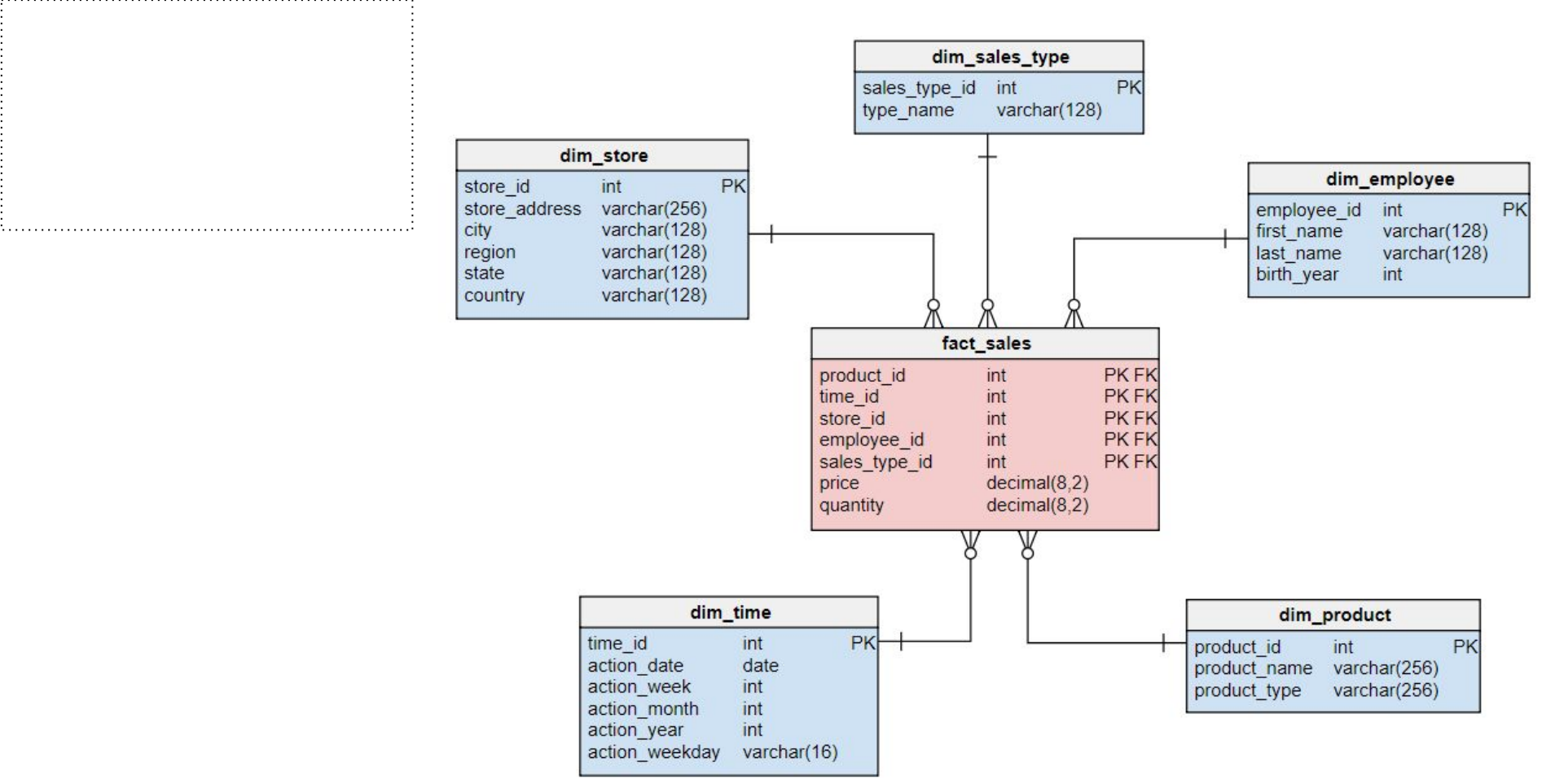


+ only need to read in relevant data

- tuple writes require multiple accesses

=>suitable for read-mostly, read-intensive, large data repositories


RDBMS
SQL Server
Flat Files
Extraction
Staging Area
Transformation
Data Warehouse
Loading


dim_sales_type
sales_type_id int PK
type_name varchar(128)
dim_store
store_id int PK
store_address varchar(256)
city varchar(128)
region varchar(128)
state varchar(128)
country varchar(128)
dim_employee
employee_id int PK
first_name varchar(128)
last_name varchar(128)
birth_year int
fact_sales
product_id int PK FK
time_id int PK FK
store_id int PK FK
employee_id int PK FK
sales_type_id int PK FK
price decimal(8,2)
quantity decimal(8,2)
dim_time
time_id int PK
action_date date
action_week int
action_month int
action_year int
action_weekday varchar(16)
dim_product
product_id int PK
product_name varchar(256)
product_type varchar(256)

Database Index

- An Index is an additional structure that is derived from the primary data. A well chosen index optimizes for reads but slows down the write. ← your use case?
- Simple database index is a Hash based Index. Some issues for an index
  - File format (encoding)
  - Deleting records
  - Crash recovery
  - Partially written records
  - Concurrency control
  - Range queries?

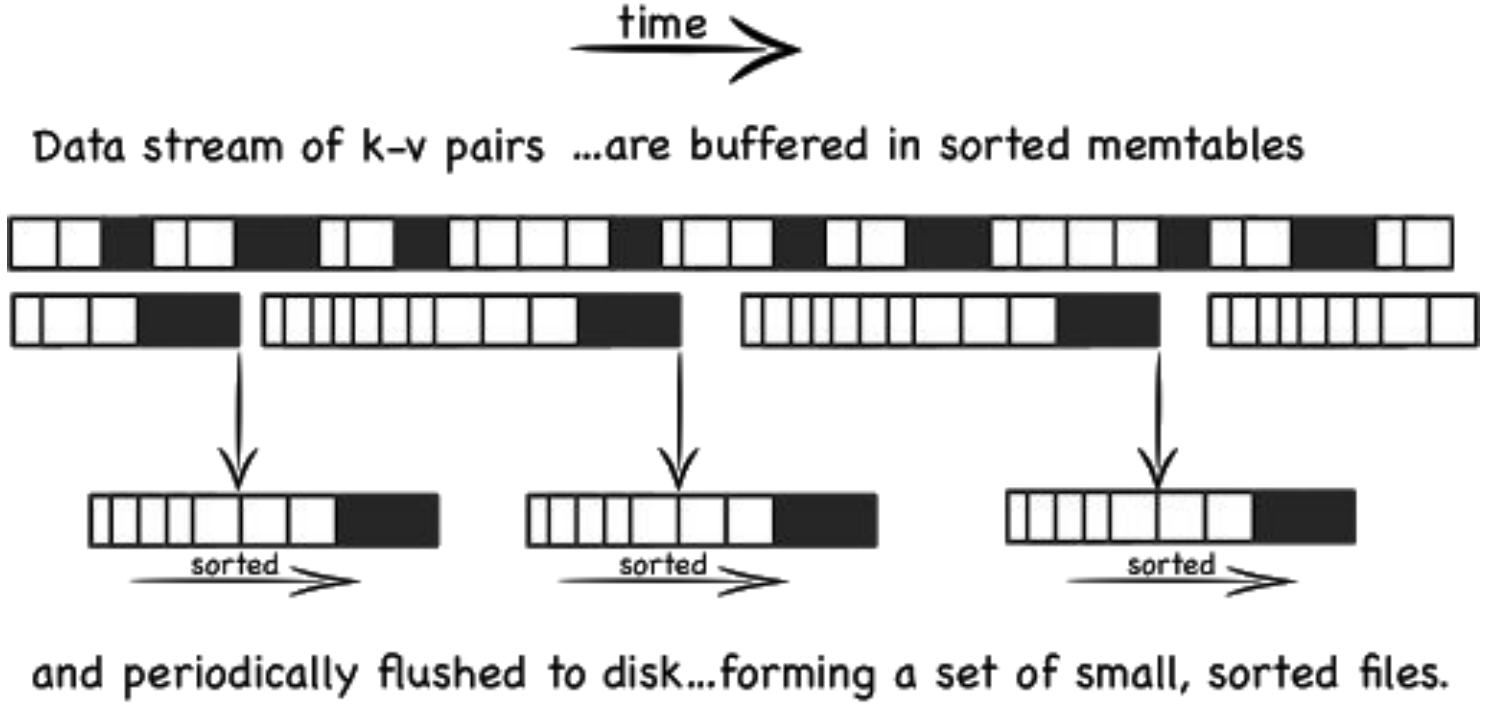
Types of storage engines

- There are two families of storage engines these databases use
  - Log-structured - LSM-Trees e.g. SSTables → HBase, Cassandra
  - Page-Oriented - B-trees. → RDBMS
- These are answers to limitations of disk access.

LSM-Trees - Log Sort Merge
SSTables - Sorted String Tables

- SSTables - in-memory mem table backed by Disk SSTable file, sorted by keys. E.g. Red-black tree or AVL Trees. Supports high wight throughput.
- Lucene - full-text search is much more complex than key-value index like SSTables. It does internally use SSTables for term dictionary.
- Bloom filters - memory-efficient data structure used for approximating the contents of a set. It can tell you if a key does not appear in the database, thus saves many unnecessary disk reads for nonexistent keys.
- Compaction is a background process of the means of throwing away duplicate keys in the log and keeping only the most recent update for each key.
  LevelDB, RocksDB, Cassandra and HBase are examples that use LSM-Trees.

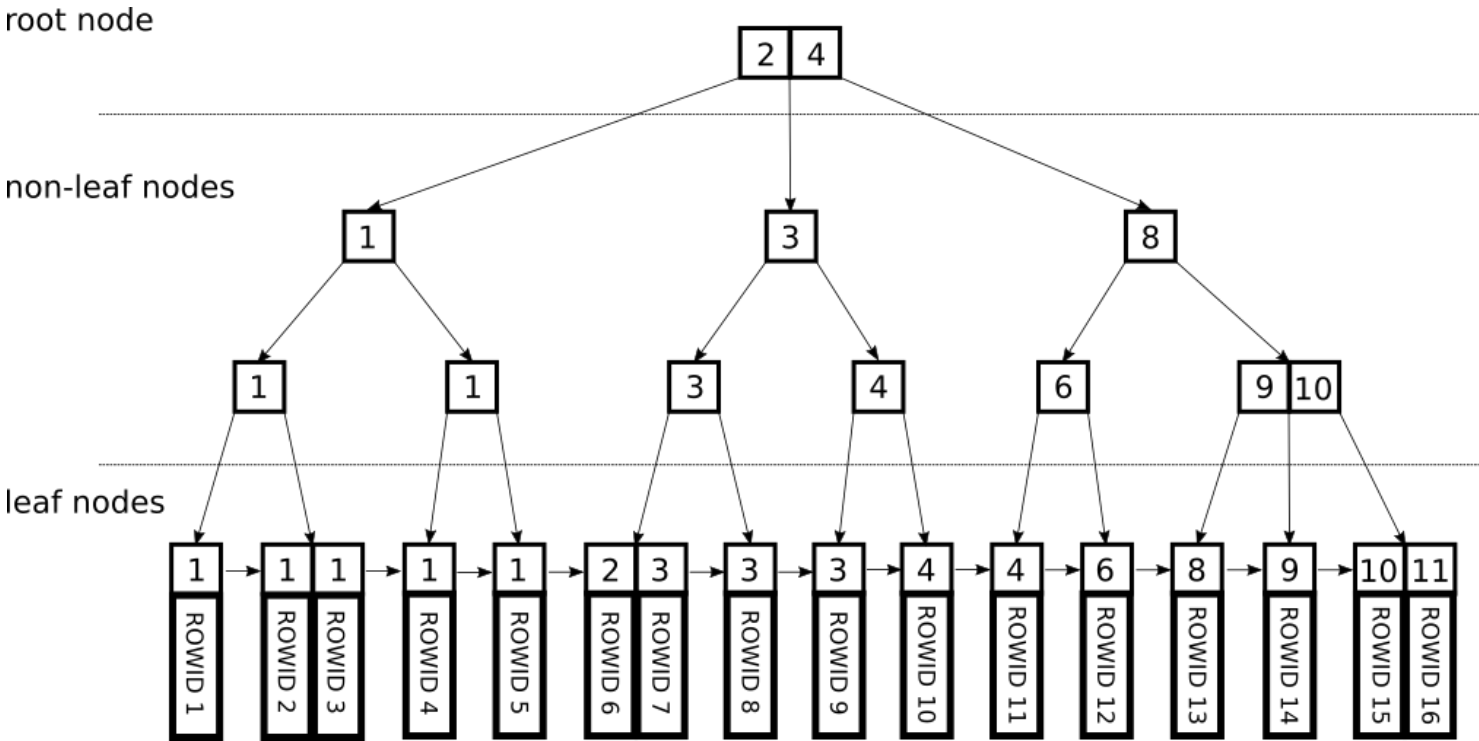
LSM-Trees - Log Sort Merge
SSTables - Sorted String Tables



B-Trees Index

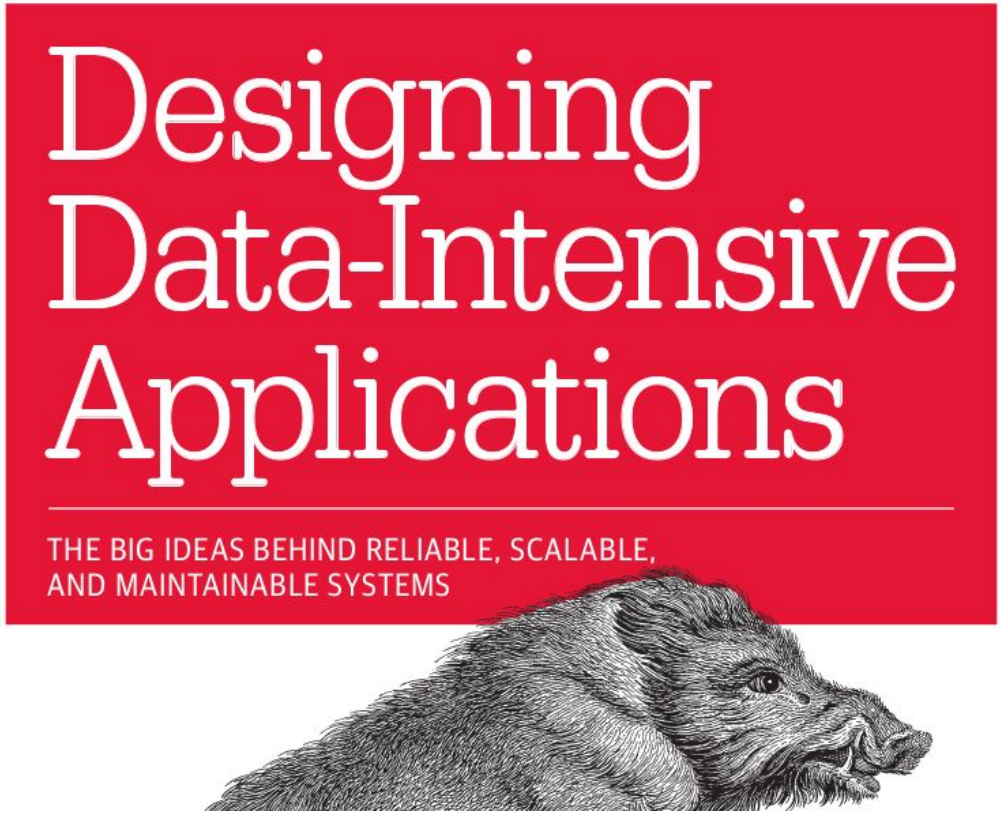
- Most widely used indexing structure is B-Trees. One place per key!
- They are the standard implementation in RDBMS and NoSQL stores today.
- It also keeps key-value sorted by keys which allows quick lookups.
- B-Trees are designed and optimized for the hardware as disks are arranged in fixed-size blocks, B-Trees also break down the data into fixed size 4KB blocks
- There is a root node and a branching factor (references to child pages).
- 4 level tree with 4KB pages with branching factor of 500 can store up to 256TB!! B-Tree is optimized for reads!
- Write ahead log is used for Crash recovery, Latches for concurrency.
- Sibling references in child node allows for easier scanning of sequential keys.

B-Trees Index



More about Index

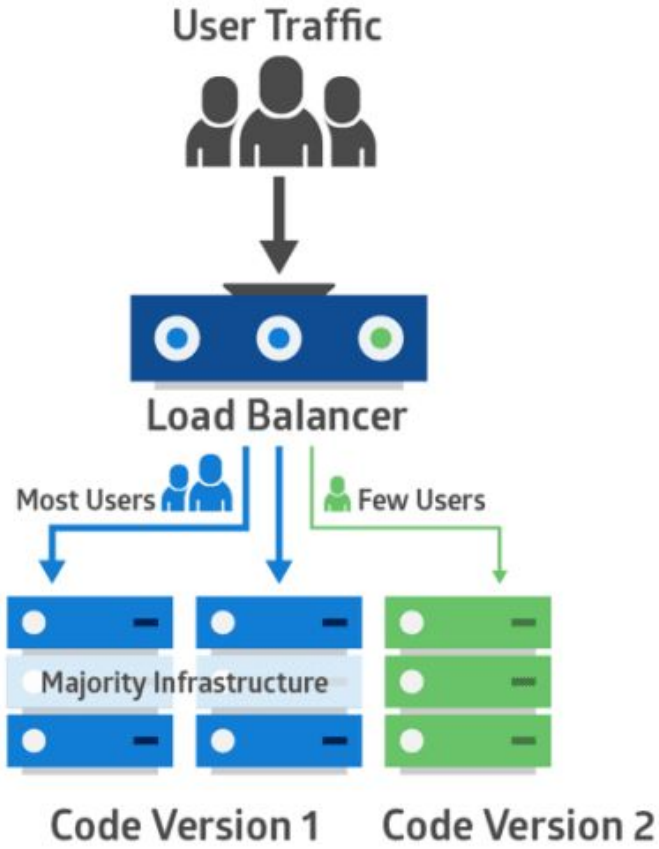
- Clustered index - inline storing of row values.
- Secondary index - helps with joins.
- Covering index - few columns are included.
- Multi-column index - multiple keys concatenated.
- Full-text search and fuzzy indexes help with spelling mistakes(edit-distances), grammar, synonyms, words near each other, linguistics etc.
- In Memory stores - VoltDB, MemSQL,Memcached.



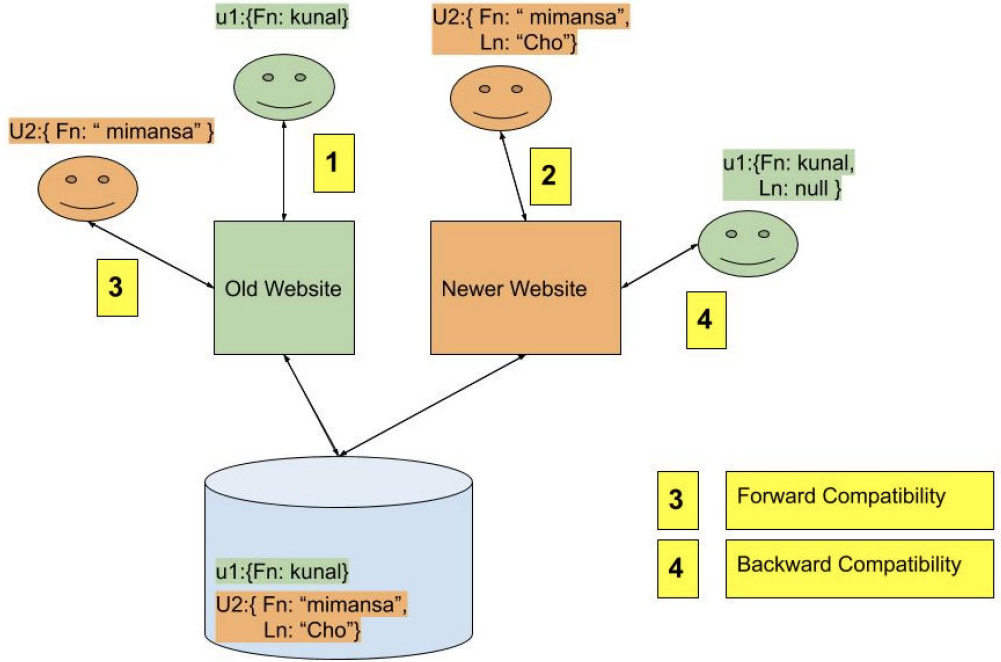
CHAPTER 4

Agile Code Evolution & Data Encoding

Rolling upgrade



Code Compatibility



Backward Compatibility: Newer code can read data that was written by older code for old and new clients.

Forward Compatibility: Older code can read data that was written by newer code for new and old clients.

Encoding/Decoding

```
{ “username” : “kunalc”,
  “favoriteNumber”: 1111,
 “Interests”: [“kayaking”, “daydreaming”]
}
```

- Encoding == Serialization == Marshalling.
- Decoding == Deserialization == Unmarshalling.

- Language specification formats - java.io.Serializable.
- Text based - JSON, CSV, XML
- Binary encoding - varying parse and size performance.

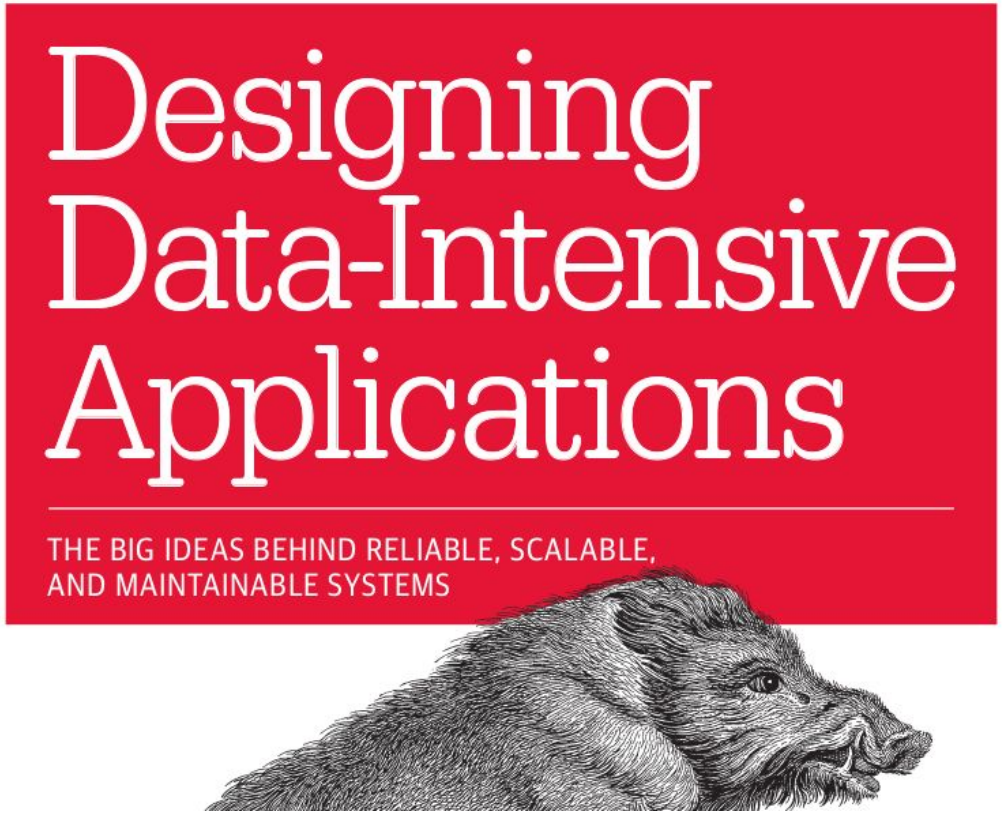
Text Json - 81 bytes, Binary JSON - 66 bytes, Thrift Binary - 59 bytes, Thrift Compat - 34 bytes, Protocol Buffer - 33 bytes, Avro - 32 bytes.

Data transfer use cases

Example data structure

{ “username” : “kunalc”,
“favoriteNumber”: 1111,
“Interests”: [“kayaking”, “daydreaming”]
}

- To a File
- To a Database
- To an external service over the network - Rest.li(json/http), SOAP/WSDL, RPC
- To an async server via message passing - one way communication/kakfa/rabbotmq/activeMQ

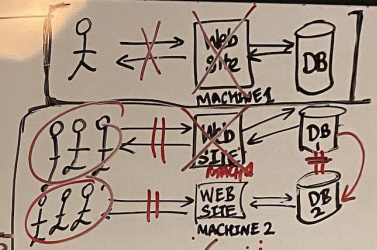


CHAPTER 5

REPLICATION

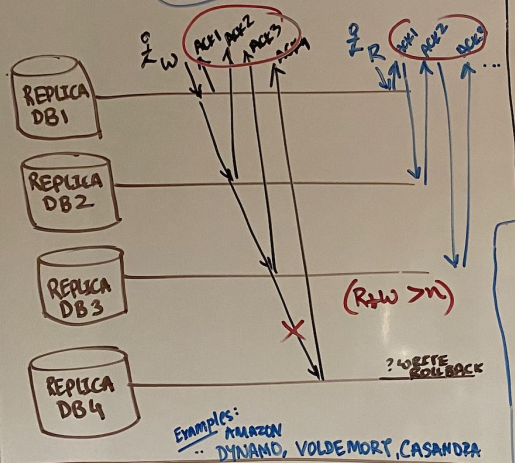
CHAPTER 5 : REPLICATION

+ MACHINE FAILURES - DB, DISK FULL
+ LATENCY FOR GLOBAL AUDIENCE
+ SCALE TO MILLIONS OF USERS
+ OFFLINE / NETWORK FAILURES



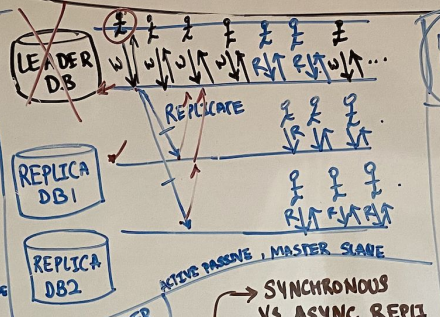
WHY REPLICATION IS NEEDED?

LEADER LESS REPLICATION 3



Examples:
.. AMAZON DYNAMO, VOLDEMORT, CASANDRA

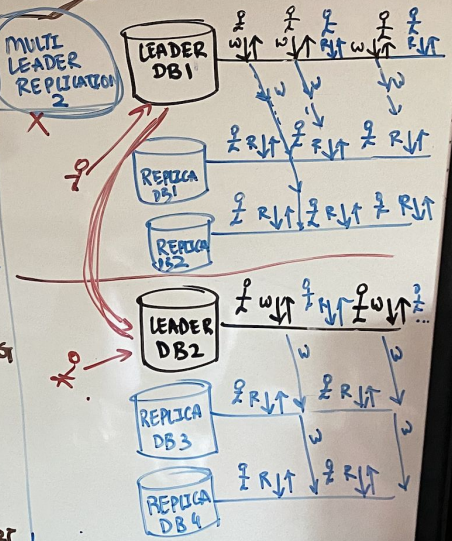
SINGLE LEADER REPLICATION 1



ACTIVE PASSIVE ; MASTER SLAVE

→ SYNCHRONOUS VS ASYNC REPLI
→ REPLICATION LAG
→ TOPOLOGY ..
→ DURABILITY VS AVAILABILITY VS LATENCY
→ FAILOVER - HEART BEAT, SNAPSHOT, COPY, CATCH UP

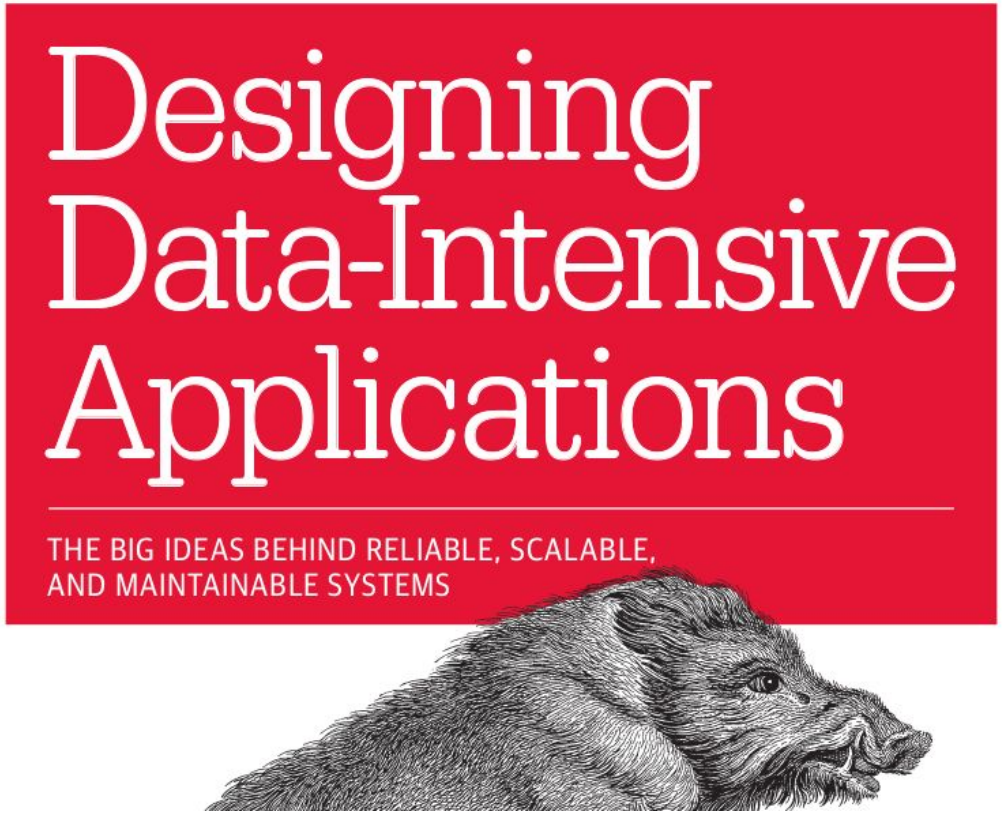
MULTI LEADER REPLICATION 2



CONFLICT RESOLUTION
MULTI DATACENTER ; QUORUM ; REPAIR

? READ AFTER WRITE CONSISTENCY - DURABILITY
? MONOTONIC READS - !TIME TRAVEL
? CONSISTENT PREFIX READS - ORDER
? ALGORITM: LAST WRITER WINS.
? EVENTUAL CONSISTENCY (CONVERGE) < MONOTONIC READS < STRONG CONSISTENCY
? GOOGLE DOCS - OPERATIONAL TRANSFORM

REPLICATION ALGORITMS
+ Statement.
+ WAL,
+ logic based
+ Trigger based

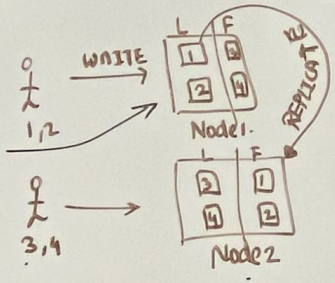


CHAPTER 6

PARTITIONING

CHAPTER-6: DATABASE PARTIONING

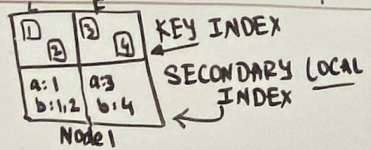
+ PARTIONING = SPLITTING = SHARDING
+ WHY? → SCALABILITY (DATA + USER + MACHINE)



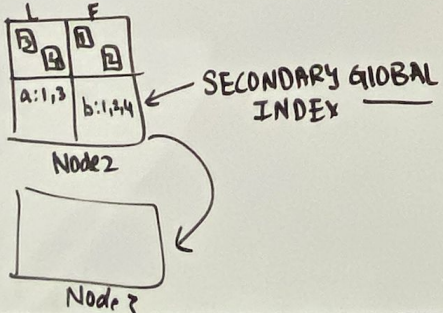
E.G.

→ 2 NODES
→ EACH 4 PARTIONS

? HOT SPOTS / SKEWS - JB
? KEY-HASH BASED PARTION

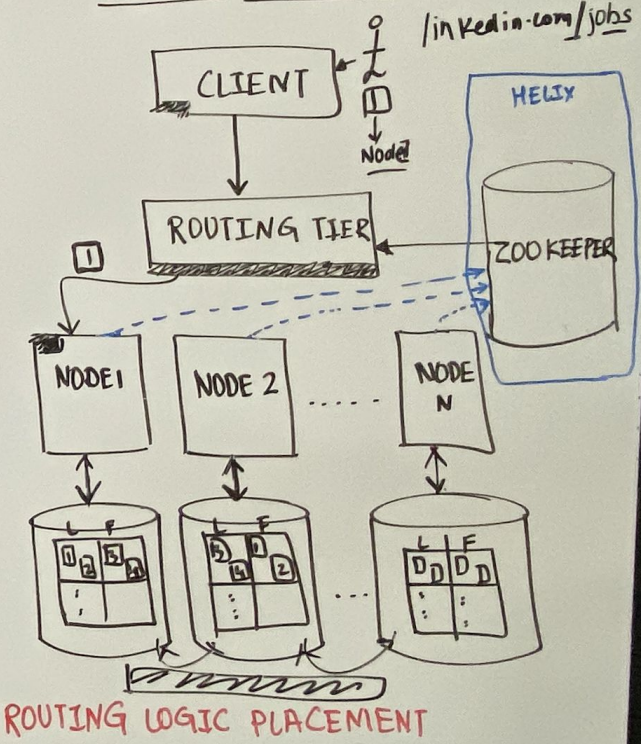


SECONDARY INDEX

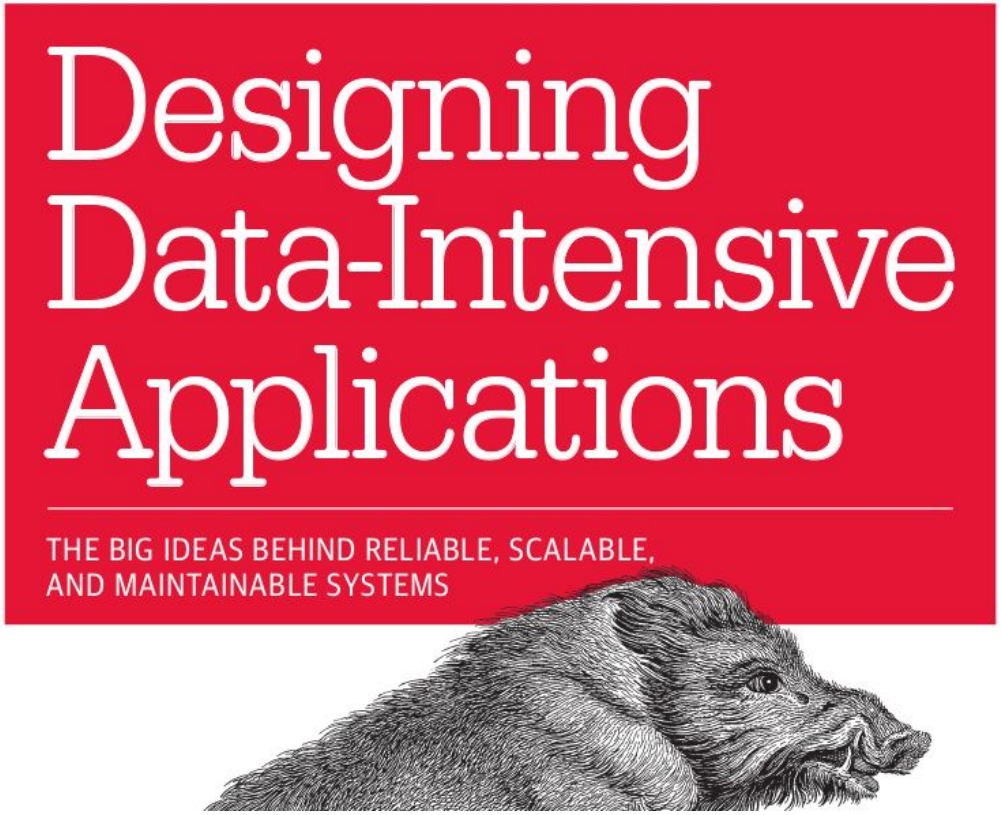


?REBALANCING STRATEGIES

REQUEST ROUTING



? ROUTING LOGIC PLACEMENT



CHAPTER 7

TRANSACTIONS

CHAPTER 7 - TRANSACTIONS

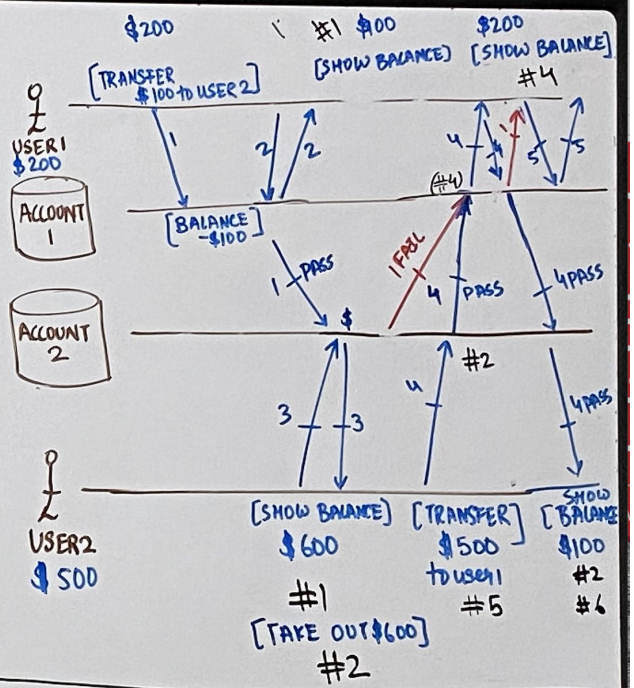
A - ATOMICITY ≠ All or NOTHING → ABORTABILITY
C - CONSISTENCY ≠ FOREIGN KEY CONSTRAINTS ...
I - ISOLATION ≠ SERIALIZABILITY
D - DURABILITY ≠ STORAGE + RECOVERY GUARANTEES

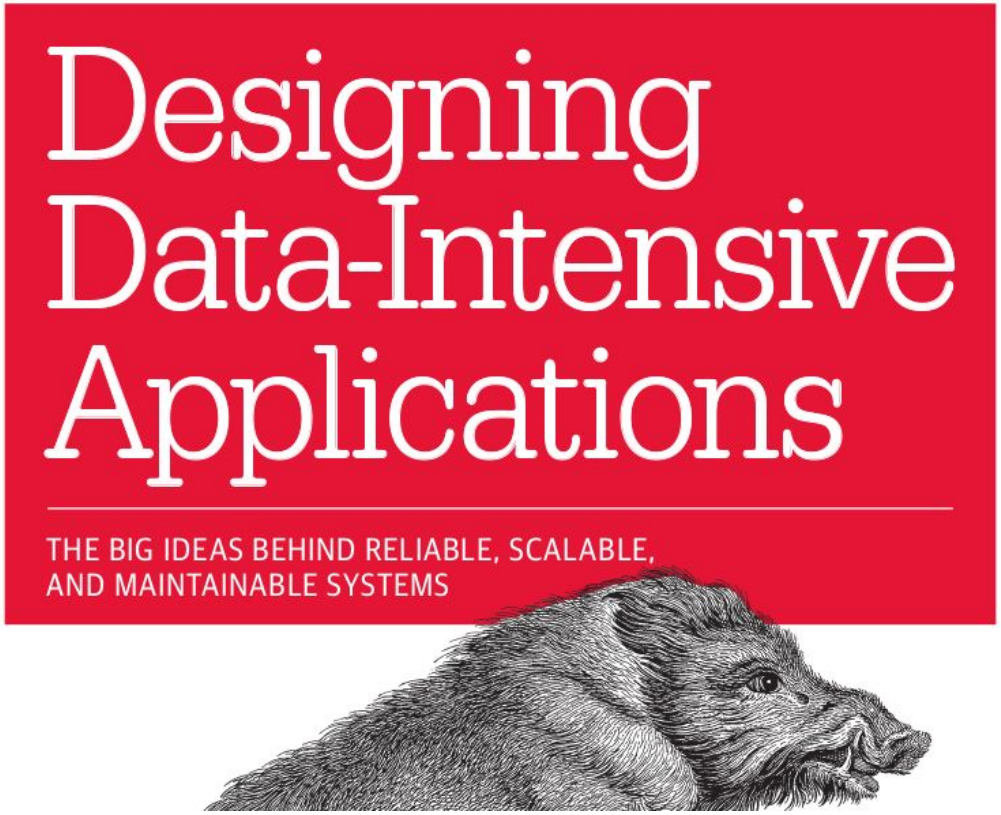
* SINGLE OBJECT WRITES VS MULTI OBJECT WRITES..
* APPLICATION DEVELOPER'S HUGE ROLE : CONCURRENCY..!!

WEAK ISOLATION LEVELS <-- ..... (II) REALITY

1. NO DIRTY READS
2. NO DIRTY WRITES
   } READ COMMITTED
3. SNAPSHOT ISOLATION - REPEATED READS
4. NO LOST UPDATES - OVERWRITE
5. NO WRITE SKEWS - PREMISE FALSE
6. NO PHANTOMS READS - ↓ READ INVALIDATED
7. SERIALIZABILITY : a) ACTUAL SERIALIZATION → ↓↑↓↑↓↑ → 1 CPU CORE 1 THREAD FOR WRITES / PARTITION
   b) TWO PHASE LOCKING → PESSIMISTIC LOCKING → LOCK & WAIT
   c) SERIALIZABLE SNAPSHOT ISOLATION → OPTIMISTIC LOCKING → ABORT IF NOT SERIAL..

(:) AHA

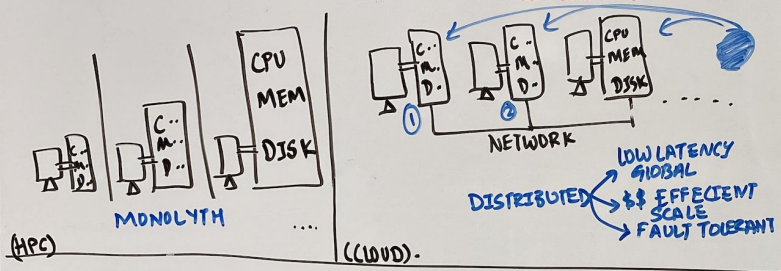




CHAPTER 8

Troubles with Distributed System

CHAPTER-8: TROUBLES WITH DISTRIBUTED SYSTEMS



* SOFTWARE IS DETERMINISTIC!! A → B → B - ALWAYS!! (SOFTWARE)
* REAL WORLD PARTIAL FAILURES.
* MONOLYTH: CUSTOM HARDWARE, DOWNTIME, CUSTOM NETWOR TOPOLOGY, ↓ ROLLING UPGRADES ! SHARED EVERYTHING, RELIABLE SLA's ↳ NO NETWORK FAULTS - SHARKS, FIRE...

* DISTRIBUTED SYSTEM: SIMULATE NETWORK OUTAGE/. RECOVERY
  - → FAULT DETECTION - TIME OUTS + All DEAD // DEAD LOCK.
  - → QUEUE'S: SWITCH/ O.S/ V.M.../
  - → NOISY NEIGHBOR → TENANCY...
  - → TCP - OPPORTUNISTIC ≃ DS

AUDIO: [16 bit @ 250 Mhz] DEDICATED

UNRELIABLE CLOCKS (1)

↑ NETWORK RT

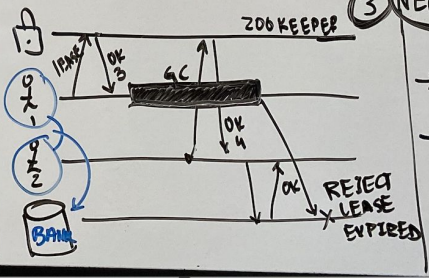
- QUARTZ CRYSTAL OSCILLATOR ← NTP SYNC
- MONOTONIC CLOCK, TIME OF DAY CLOCK
  System.nanoTime() , System.getCurrentTime()..
- CLOCK DRIFTS: LEAP SECONDS
- PRECISION TIME PROTOCOL - GPS RECEIVERS
- SILENT & SUBTLE DATA LOSS - LRW - ORDERING
- LOGICAL CLOCK - GOOGLE SPANNER - ORDERING RANGE (AZURE COSMOS) CONFIDENCE INT..

PROCESS PAUSE (2)

DETECT → RECOVER FAILURES

- STOP THE WORLD - GC PAUSE → GC TUNING → HOTSPOT
- VM MIGRATION - VM PAUSE + I/O PAUSE / PAGING
- SIGSTOP, CTL-Z (!!)
- RTOS: STRICT LIB EXEC TIME, NO DYNAMIC MEMORY - LOW THROUGHPUT
- QUORUM: HEART BEAT / VOTING

(3) NETWORK



- → BYZANTINE FAULTS
- → CORRECT ALGO SECURE FENCE SAFE, LIVE

CS → SE

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS

CHAPTER 9

CONSISTENCY & CONSENSUS

CHAPTER 9 - CONSISTENCY AND CONSENSUS - (ABSTRACTION: ONE COPY OF DATA)

① EXAMPLES

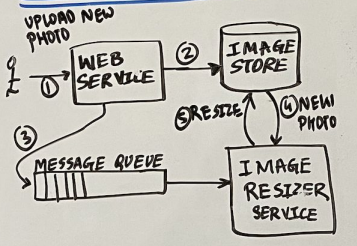
→ HUSBAND & WIFE TAKE MONEY OUT FROM SAME ACCOUNT -ve $$
→ NO DOCTOR ONCALL HOSPITAL
→ FOOTBAL RESULTS DIFFERENCE
→ OUT OF ORDER MESSAGES..
→ SAME SEAT ASSIGNED TO MULTIPLE PEOPLE ON FLIGHT

② REAL LIFE

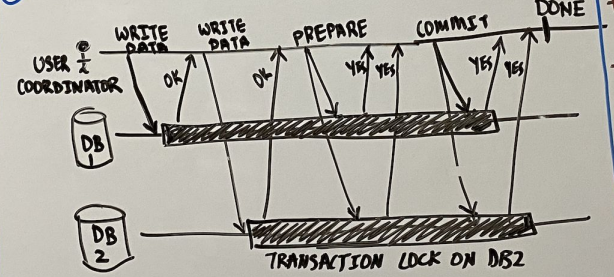
RAFT ZAB VSR PAXOS

+APACHE ZOOKEEPER
- LINEARIZABLE GUARANTEE (PAXOS) - LEADER ELECTION
- ORDERING → zxid
- FAILURE DETECTION +HEART BEAT
- CHANGE NOTIFICATION
- NODE WORK PARTIONING
- CONFIG MANAGEMENT
- SERVICE DISCOVERY..

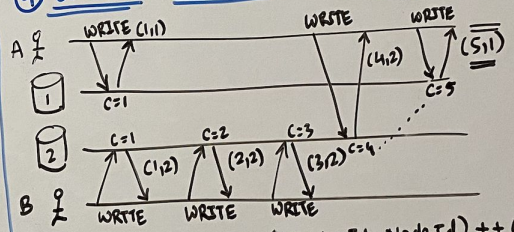
② LINERIAZABILITY



③ 2PC = TWO PHASE COMMITS



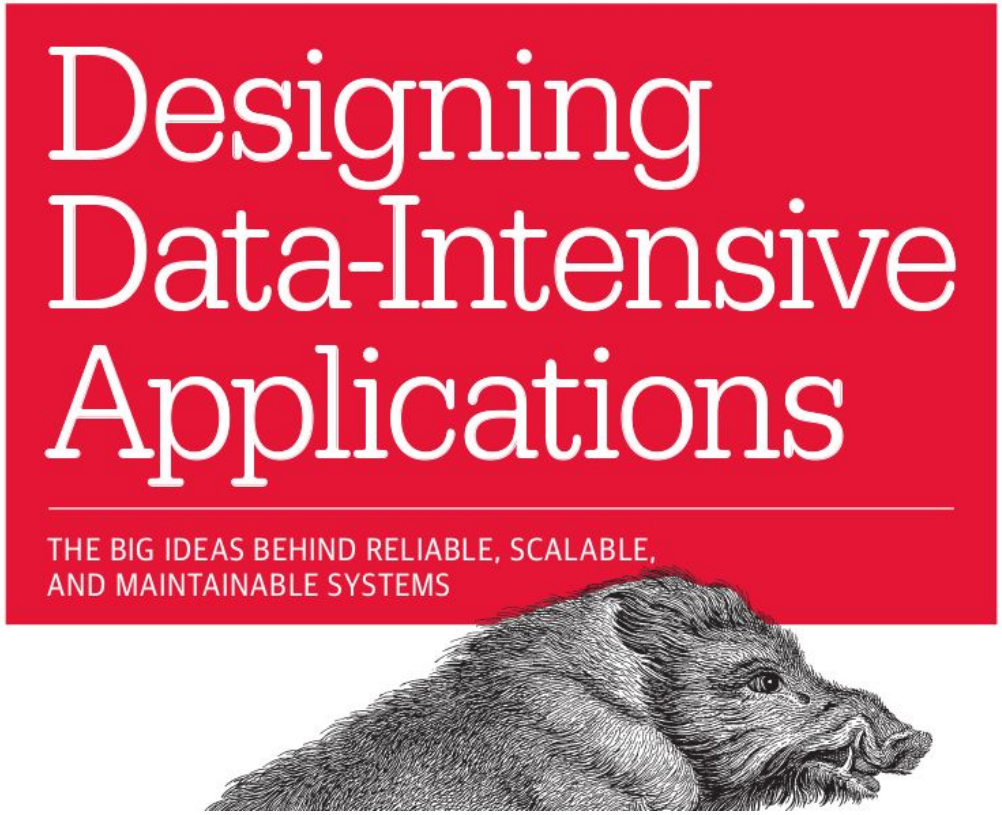
④ LAMPORT TIMESTAMP & TOTAL ORDER BROADCAST



(counterId, NodeId) ++ Catchup

⑤ OTHER TOPICS IN THIS CHAPTER

+EVENTUAL CONSISTENCY == CONVERGENCE
+NETWOR PARTION SIMULATE
+↑STRONGER GUARANTEES
↓LOW TROUGHPUT
↓HIGH LATENCY
+FIDELITY: READ YOUR OWN WRITES
+SPLIT BRAIN
+CAUSALITY << LINEARIZABLE TOTAL ORDER.
+NO OF FAILURES (10X SLOW)
↓HUMAN INTERVENTION



CHAPTER 10

BATCH PROCESSING

CHAPTER 10: BATCH PROCESSING : SUMMARY / INSIGHTS / A/B TEST / BUILDIND INDEX / (K,V STORE) PYMK (11)

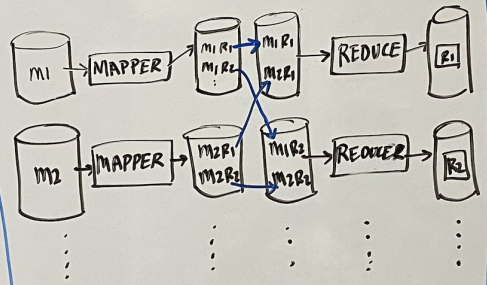
① UNIX BATCH PROCESSING

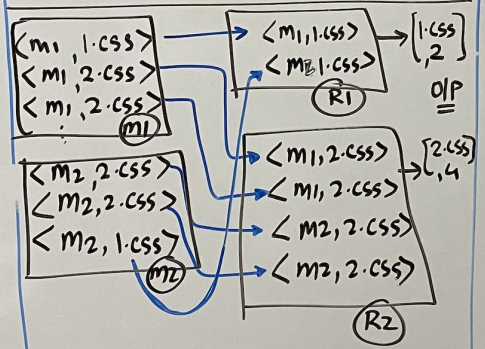
```
Cat /var/log/access.log
| awk '{print $6}'
| Sort
| Uniq -c
| Sort -r -n
| head -n 5
```

```
→ 126.1.5.3 - 123 [27/02/20]
"GET /css/1.css HTTP/1.1"
200 3344 "abc.com"
"Mozilla.... Chrome/40..."
```

UNIFORM, LOGIC SEPARATE, ITERATE/EXPERIMENT

② MAP REDUCE





JOIN ALGO: RERUN (FAULT TOLERANT), BROADCAST HASH, PARTITION HASH.. HCATALOG, AZKABAN, LOCALIZED COMPUTE, HOT KEYS

10K MACHINES, PETABYTES, FAULT TOLERANT

③ BEYOND MAP REDUCE

MPP → MAP REDUCE → PIG/HIVE → SPARK (writing..) → GRAPH/ITERATIVE..

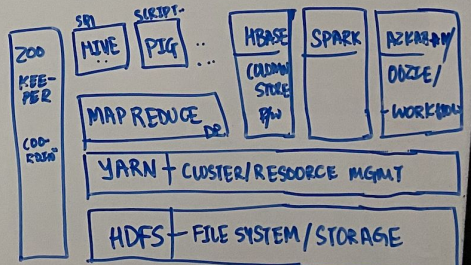
MPP (TERADATA): STORAGE, PROCESSING, FAULT TOLERANCE, +5% CHANCE ↑

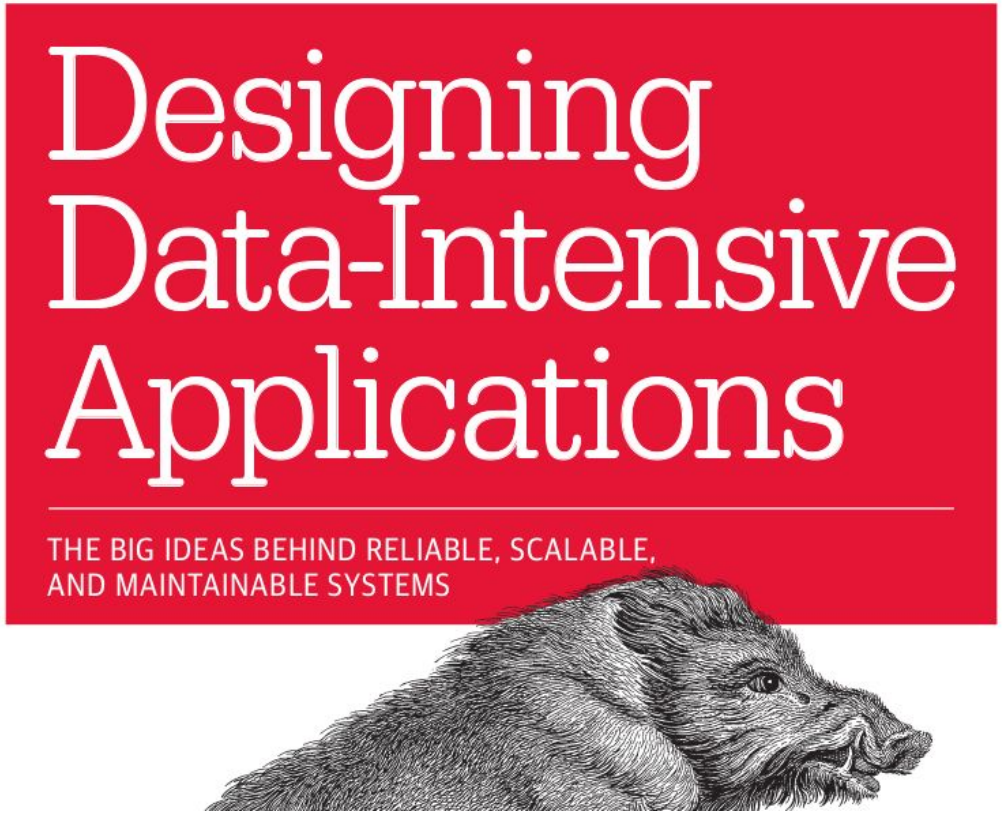
PIG/HIVE: + DATA FLOW, + PROGRAMMER, + ANALYTICAL SQL (HIVE)

SPARK:
+ FASTER - NO CHAINING / STREAMING
+ NO UNECESSARY MATERIALIZATION INTERMEDIATE STATE
+ OPERATORS (EXECUTION)..
+ ENTIRE ENGINE AS ONE JOB.
+ SKIPS SORTS WHEN NOT NEEDED
+ IN MEMORY WHEN POSSIBLE NO REPLICATION OVERHEAD
+ NO NEW JVM PER TASK
+ RDD - RESILIENT DISTRIBUTED DATASET

SCALE → PROGRAM MODEL → EFFECIENCY (*) → EXPAND USECASE (MAHOOT..)

HADOOP ECOSYSTEM : QUICK INTRO..





CHAPTER 11

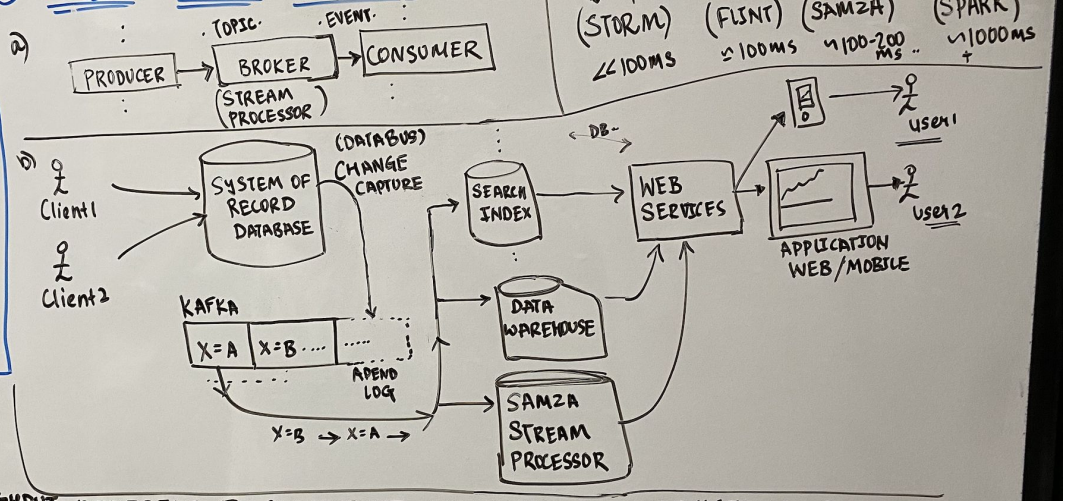
STREAM PROCESSING

CHAPTER 11: STREAM PROCESSING

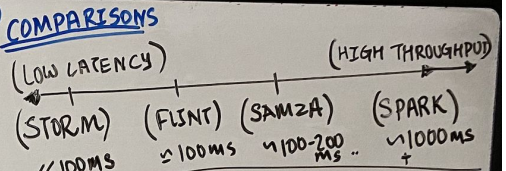
① USE CASES (UNBOUNDED + REALTIME)

1. FRAUD DETECTION: CREDIT CARD LOST
2. ANALYTICS: TRENDS.
3. ANOMALY DETECTION: SOFTWARE/BUG
4. STOCK MARKET TRADES: ORDER EXECUTION
5. ATTACK DETECTION....

② STREAM SYSTEM ARCHITECTURE

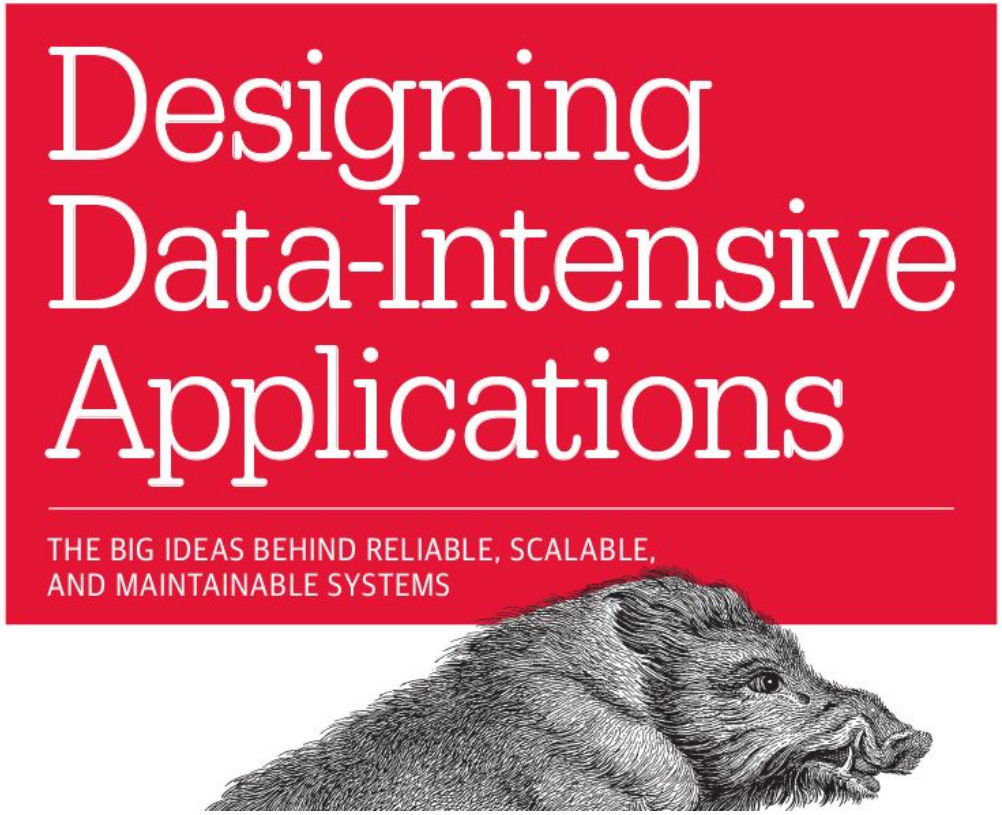


③ COMPARISONS



④ IMPORTANT CONCEPTS

* COMPARISONS: LATENCY, THROUGHPUT, ORDERING, BACKPRESSURE, ELASTICITY, STATE MANAGEMENT, STRICTNESS DELIVERY...
* POLLING VS NOTIFICATION → DB TRIGGER * BROKER RESPONSIBLE FOR DURABILITY. / RETENTION (DATA)
* STREAMING ALLOWS FOR MULTIPLE VIEWS OF THE SAME DATA. → A/B TEST → NO SCHEMA EVOLUTION....
* DATABASE IS THE CACHE SUBSET OF LOG REPRESENTATION (LATEST VALUE) → CART ITEMS REMOVED DATA
* WINDOW CONCEPT: TUMBLING WINDOW, HOPPING WINDOW, SLIDING WINDOW, SESSION WINDOW..
* MICRO BATCHING, IDEMPOTENT WRITES, CHECKPOINTING, TRANSACTIONS, FAULT TOLERENCE..



CHAPTER 12

FUTURE of DATA SYSTEMS

CHAPTER-12 FUTURE OF DATA SYSTEMS

*) HOW THINGS ARE TODAY ==> HOW THINGS SHOULD BE IN FUTURE

ZOOM OUT - TIME TO TIME → RE DESIGN

RELIABLE ←→ SCALABLE ←→ MAINTAINABLE
(CORRECT) (ROBUST) (EVOLVABLE)

*) BE ABLE TO READ BETWEEN THE LINES - SYSTEM CHOICES → IS THIS ETHICAL → BENEFITS HUMANITY??

* TOTAL ORDERING IS HARD & COSTLY:
STATELESS MICRO SERVICES, MULTIPLE DATACENTER LEADERS
STATEFUL CLIENTS / OFFLINE SUPPORT

*) CAUSAL RELATIONSHIPS: Email sends, GROUP SUB, CONSTRAINTS (BANK BAL$)
↑ (FOCUS 1 THING)

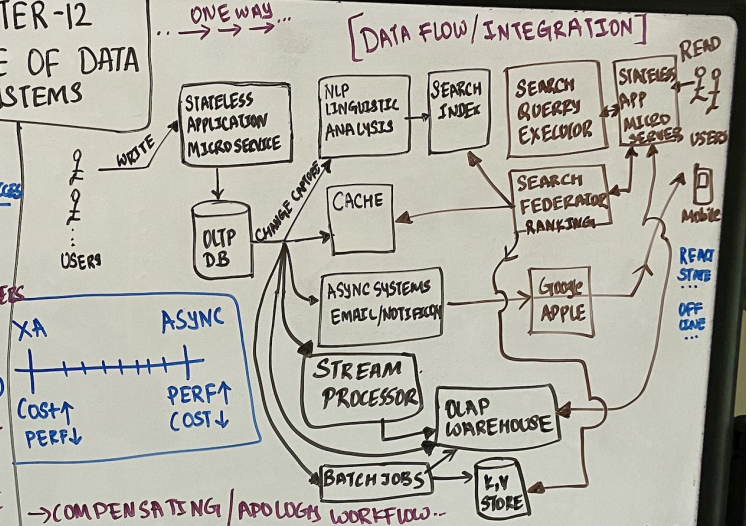
*) UNBUNDLING DATABASES: COMPLEX MAINTANENCE..
OVERWRITE BATCH / STREAM O/P → WINDOW / SYNC..
SNAPSHOT → INDEX FILTER → CREATE → PATCH → MERGE
SAME AS DB REPLICA SETUP / VS INDEX SETUP..

*) ORDERED LOG EVENTS + IDEMPOTENT CONSUMER: VS
↳ END TO END REQUEST ID, DE DUPE AT CONSUMER..

→ (R1) - DEBIT $10 FROM A, CREDIT $10 TO B — LOG
(R1,D1) → DEBIT $10 , (R1,C1) → CREDIT $10 → CRASH / REPLAY
↓ DONE DONT REDO .. ↓ NOT DONE → FINISH..] CONSUMER dedupe R1..

→ EVENTUAL CONSISTENCY (DELAY) IS OK - TIMELINESS
INTEGRITY (DATA LOSS / CORRECTNESS) GAPS NOT OK..

[DATA FLOW / INTEGRATION]



→ COMPENSATING / APOLOGY WORKFLOW..

→ TRUST BUT VERIFY - EVEN BATTLE TESTED CODE FAILS - MEM CORR-

→ ENABLE CONTINUOUS AUDITING / DEBUG / HEAL CAPABILITY..
END TO END CHECKS.

DO THE RIGHT THING

→ PREDICTIVE ANALYTICS MERELY CODIFY THE PAST... / PEOPLE /..

→ BIAS / ECHO CHAMBERS / SYSTEMS THINKING - UNINTENDED CONSEQ...

PRIVACY & TRACKING !=! SURVEILLANCE...
(EAVESDROPPING) (POLLUTION)

Introduction to CDN