

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



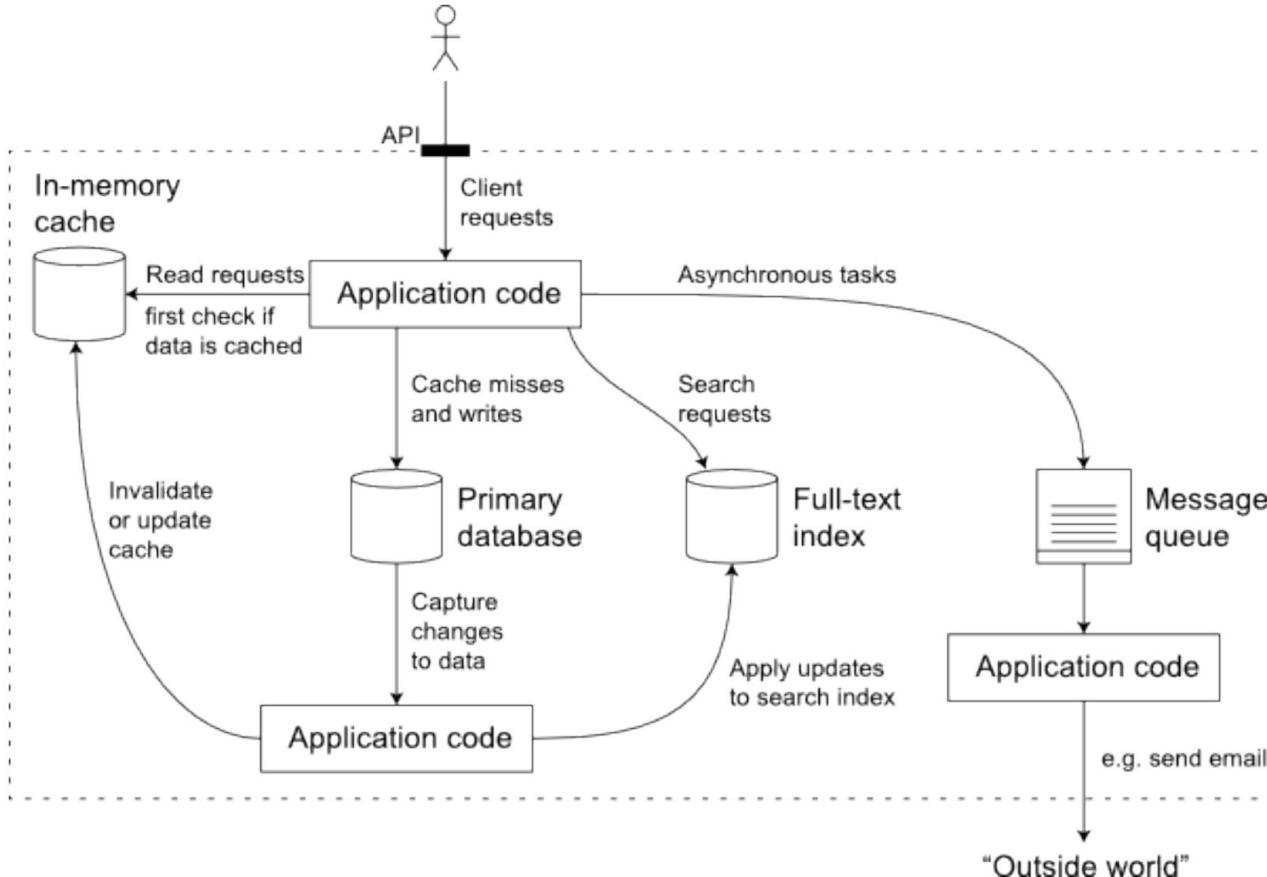
CHAPTER 1

**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/24f14e3b30012c2bc7e3abbd16e2b3365d6f920>

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

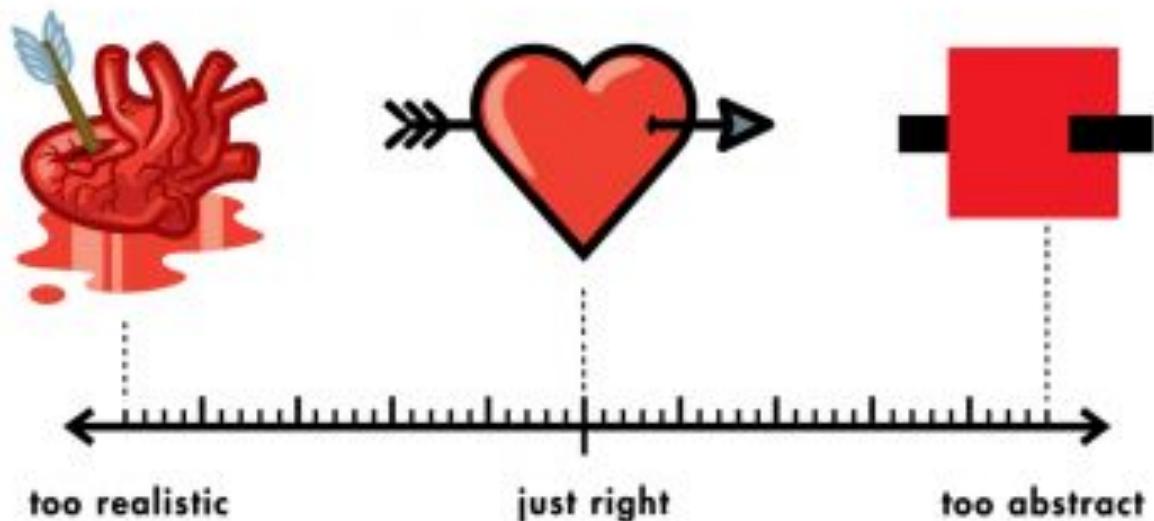
THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS

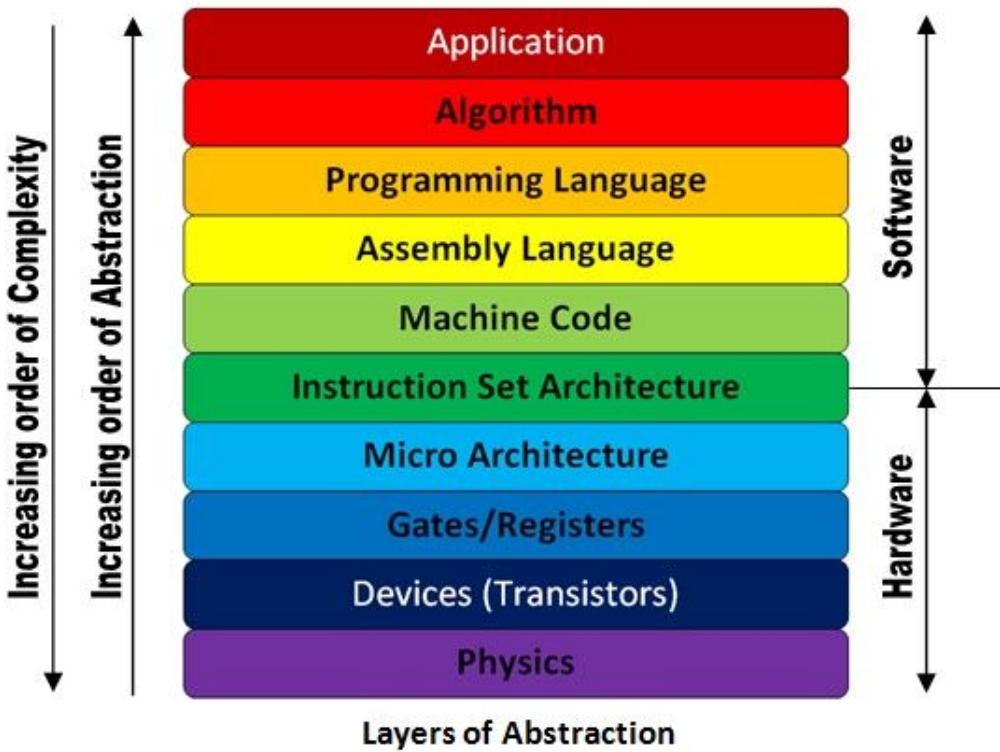


CHAPTER 2

DATA MODELS & QUERY LANGUAGES

THE ABSTRACT-O-METER





Data Abstraction



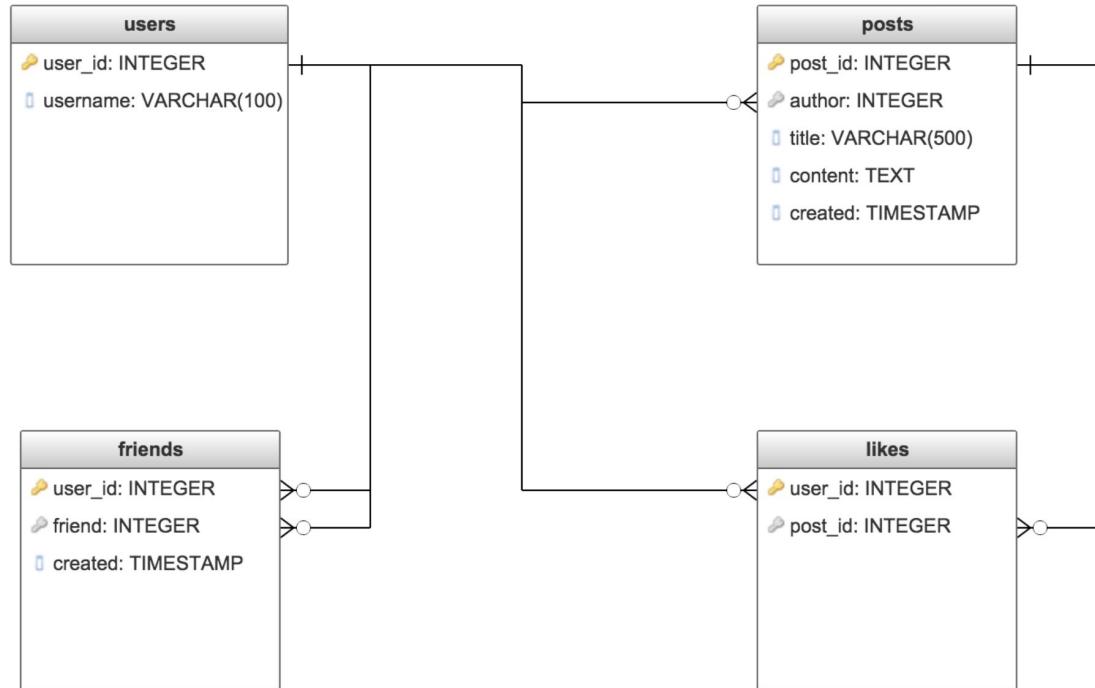


Model a social network
Users -> Friends -> Posts -> Likes



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 model

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
```

Users

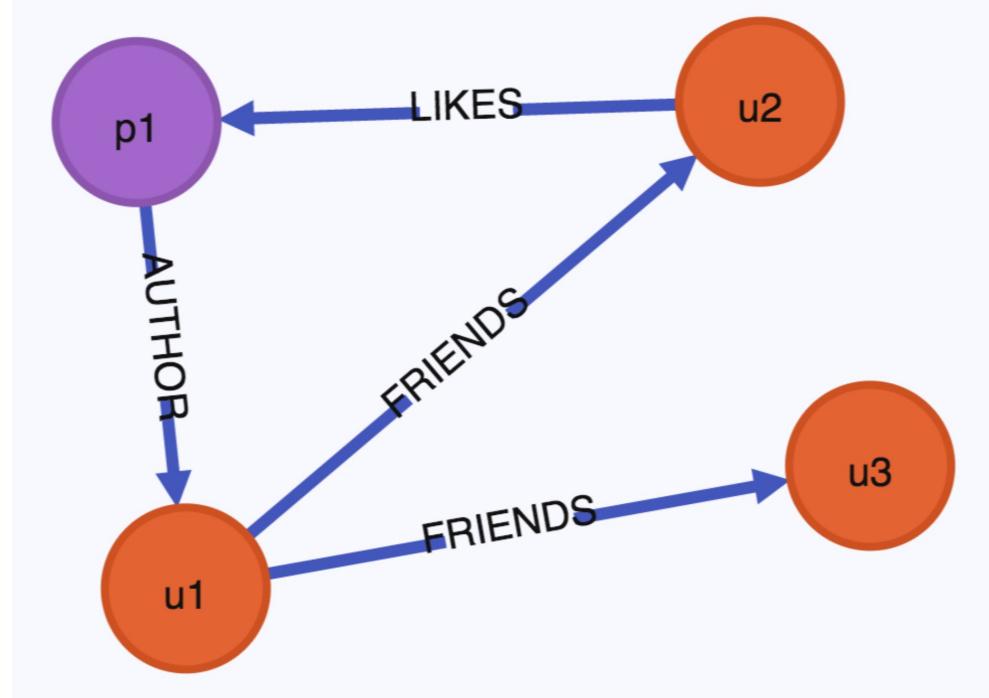
```
{  
  "user_id": "u1",  
  "username": "grahamcox",  
  "friends": {  
    "u2": "2017-04-25T06:41:11Z",  
    "u3": "2017-04-25T06:41:11Z"  
  }  
}
```

Document based model

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	<ul style="list-style-type: none">• Optimized for Transaction and batch processing (read throughput), joins..• Data organized as table/relations• Object Relational Mapping needed.• Oracle, MySQL, PostgreSQL..
02	Document Database	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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®



PostgreSQL



Couchbase



neo4j



ArangoDB

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



CHAPTER 3

STORAGE & RETRIEVAL

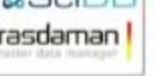
Storage & Retrieval



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

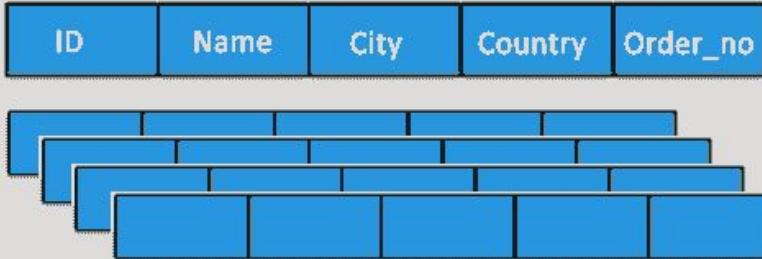
Data Analysis & Platforms 	Databases / Data warehousing 	Operational 	Multivalue database 
Business Intelligence 	Data Mining 	Social 	Big Data search 
KeyValue 	Document Store 	Graphs 	Multidimensional 
Object databases 	Multimodel 	XML Databases 	

Created by www.BigData-Start.com

Which database to use ??

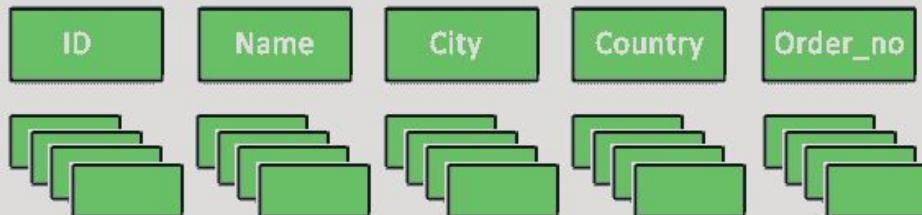
- There are two broad categories of databases - OLTP and OLAP each with different read pattern, write patterns, user using it, data size etc.
 - OLTP - Online Transaction processing database are optimized for latency.
 - E.g. MySQL, Oracle.
 - OLAP - Online Analytical processing databases are optimized 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



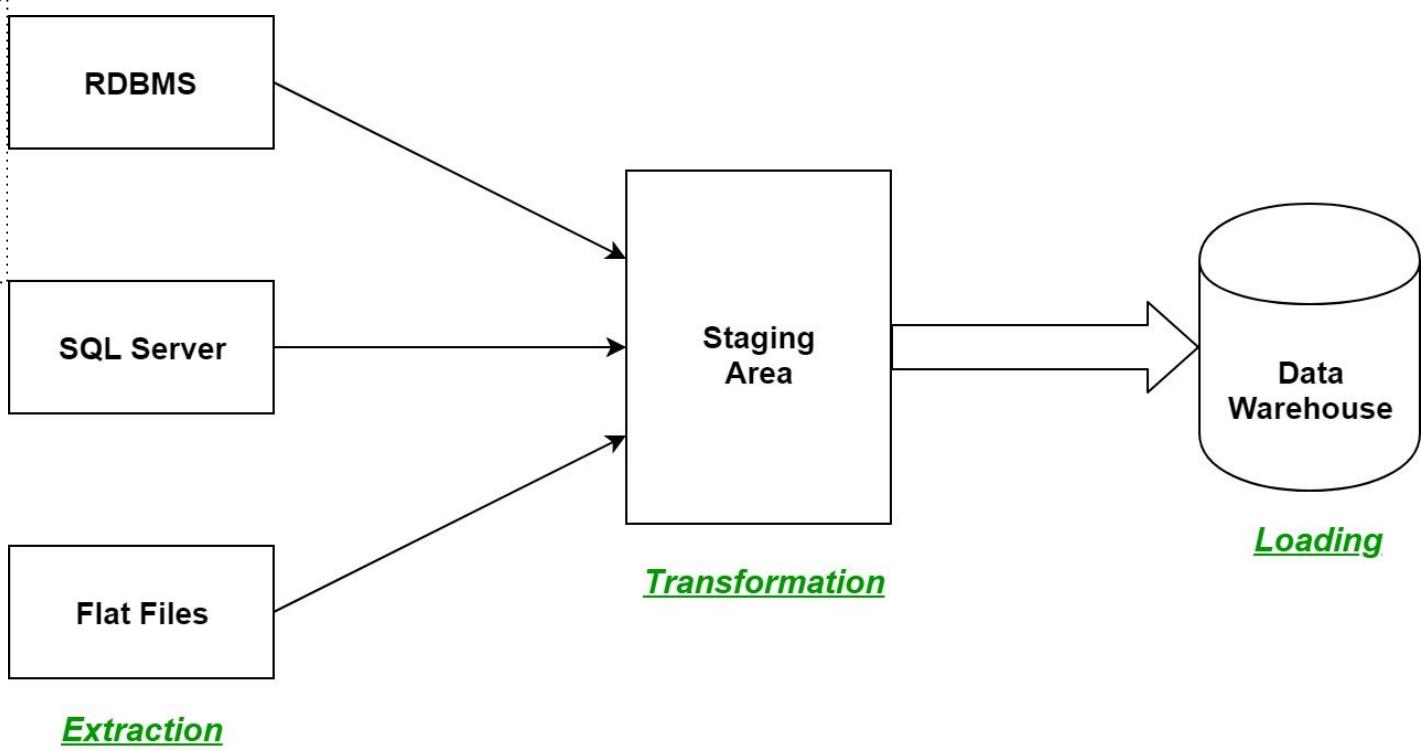
- + easy to add/modify a record
- might read in unnecessary data

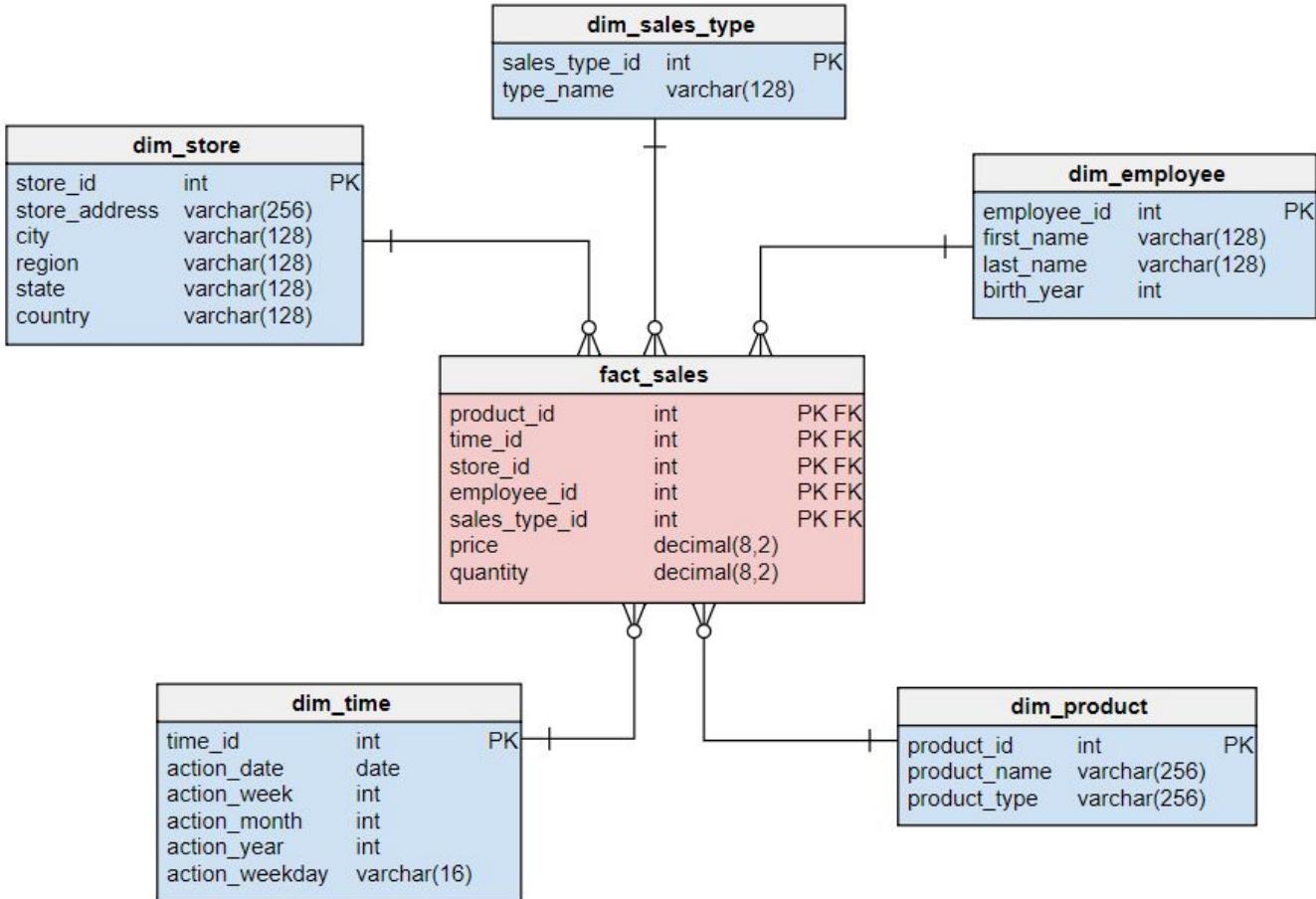
column-store



- + only need to read in relevant data
- tuple writes require multiple accesses

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





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?

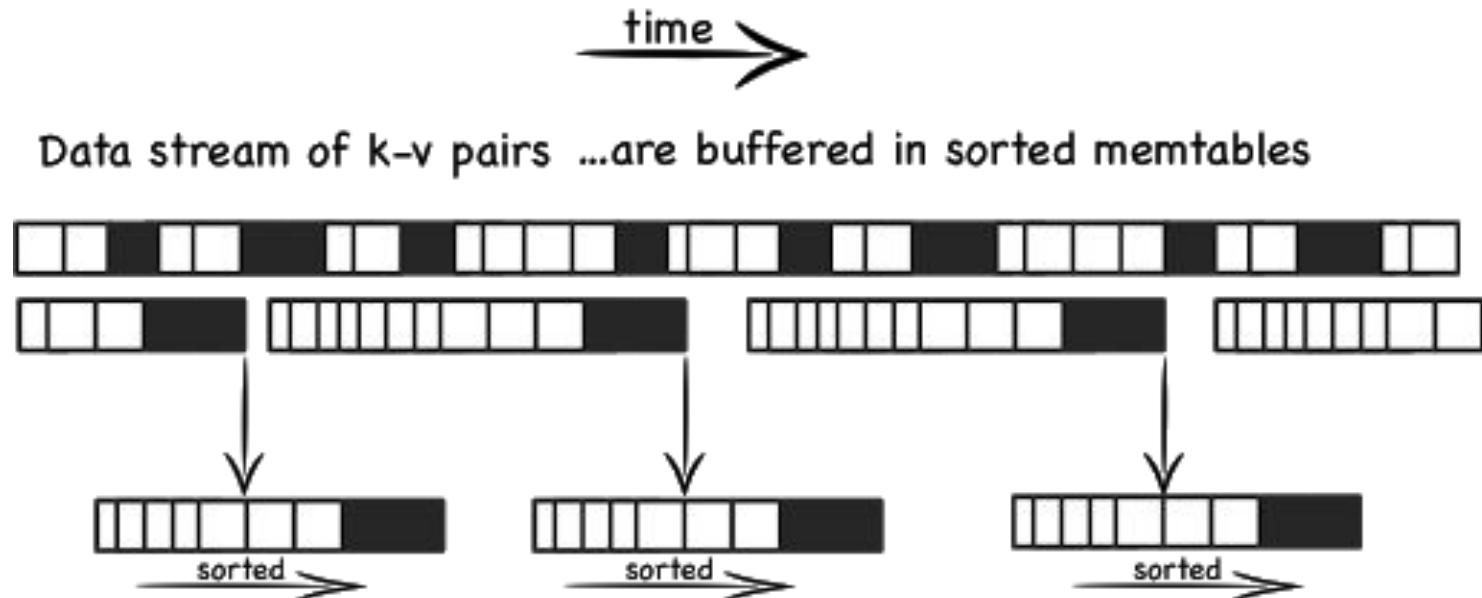
Types of storage engines

- 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

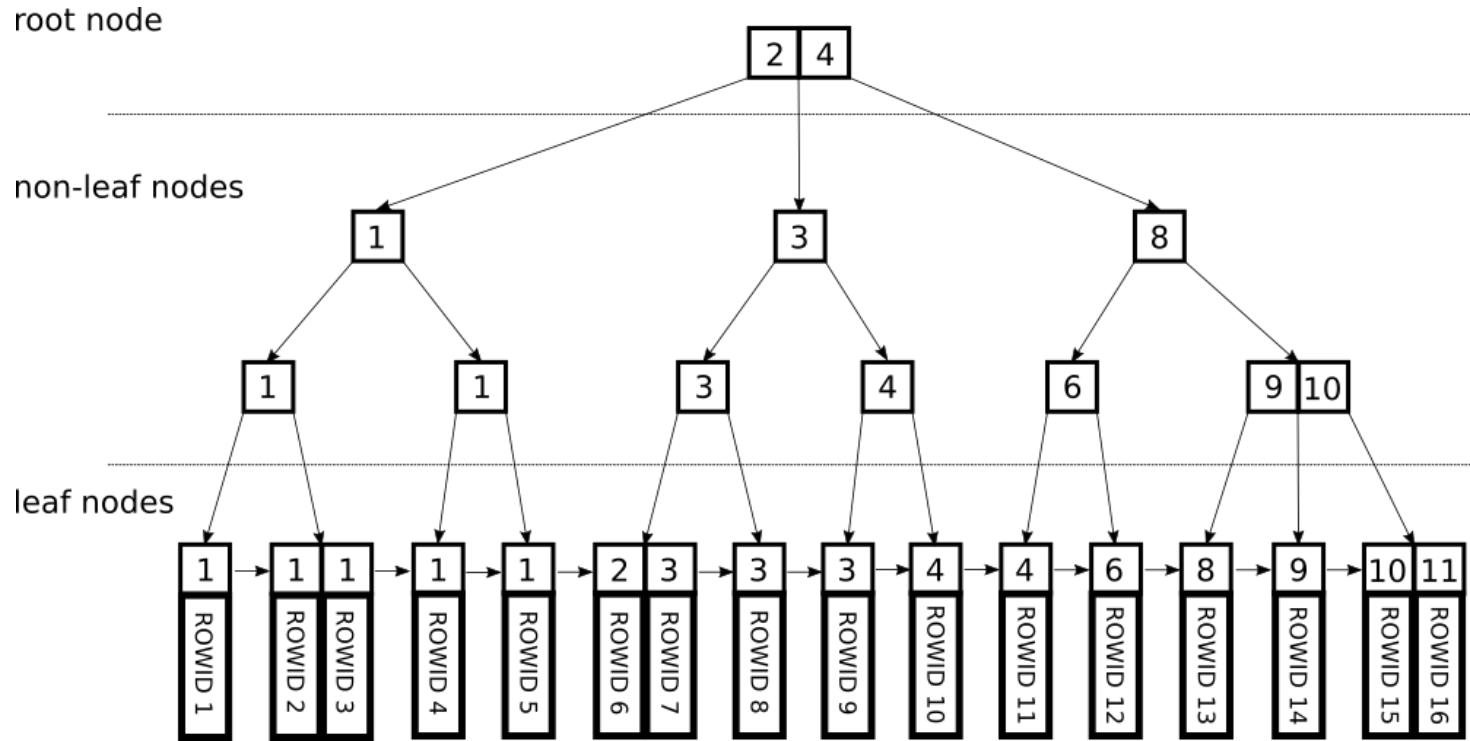


and periodically flushed to disk...forming a set of small, sorted files.

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.

Designing Data-Intensive Applications

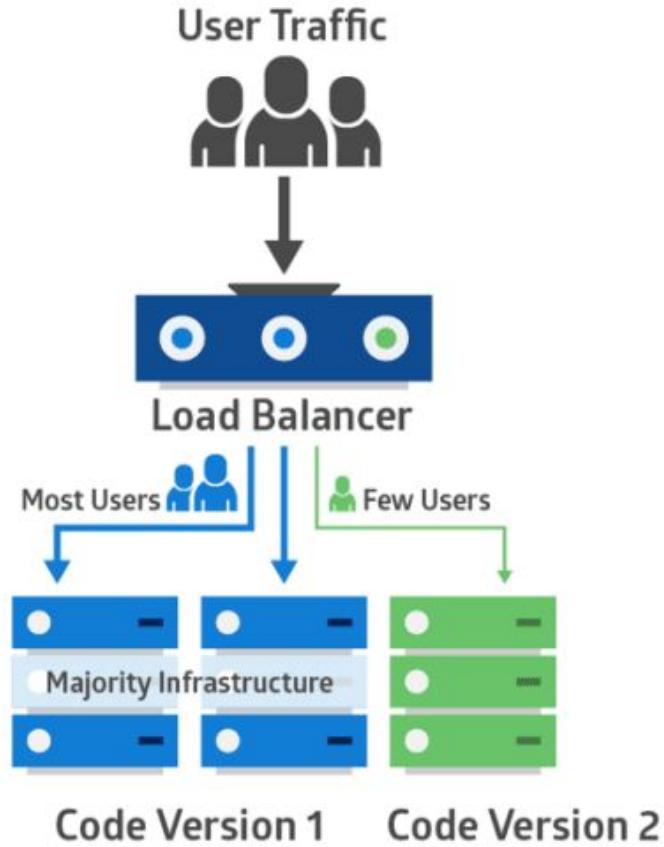
THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



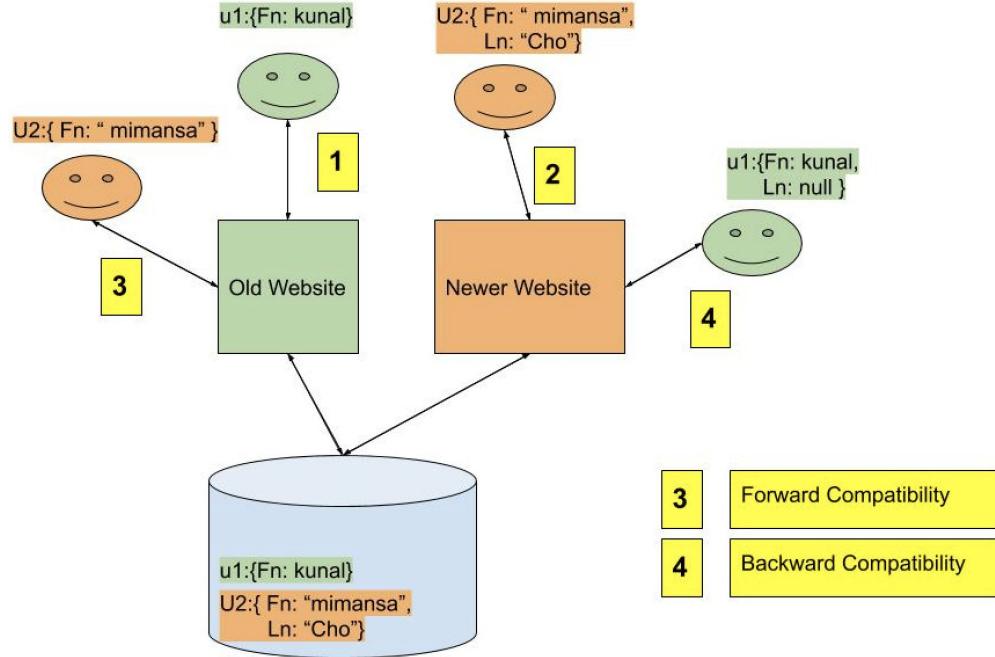
CHAPTER 4

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/kafka/rabbitmq/activeMQ

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



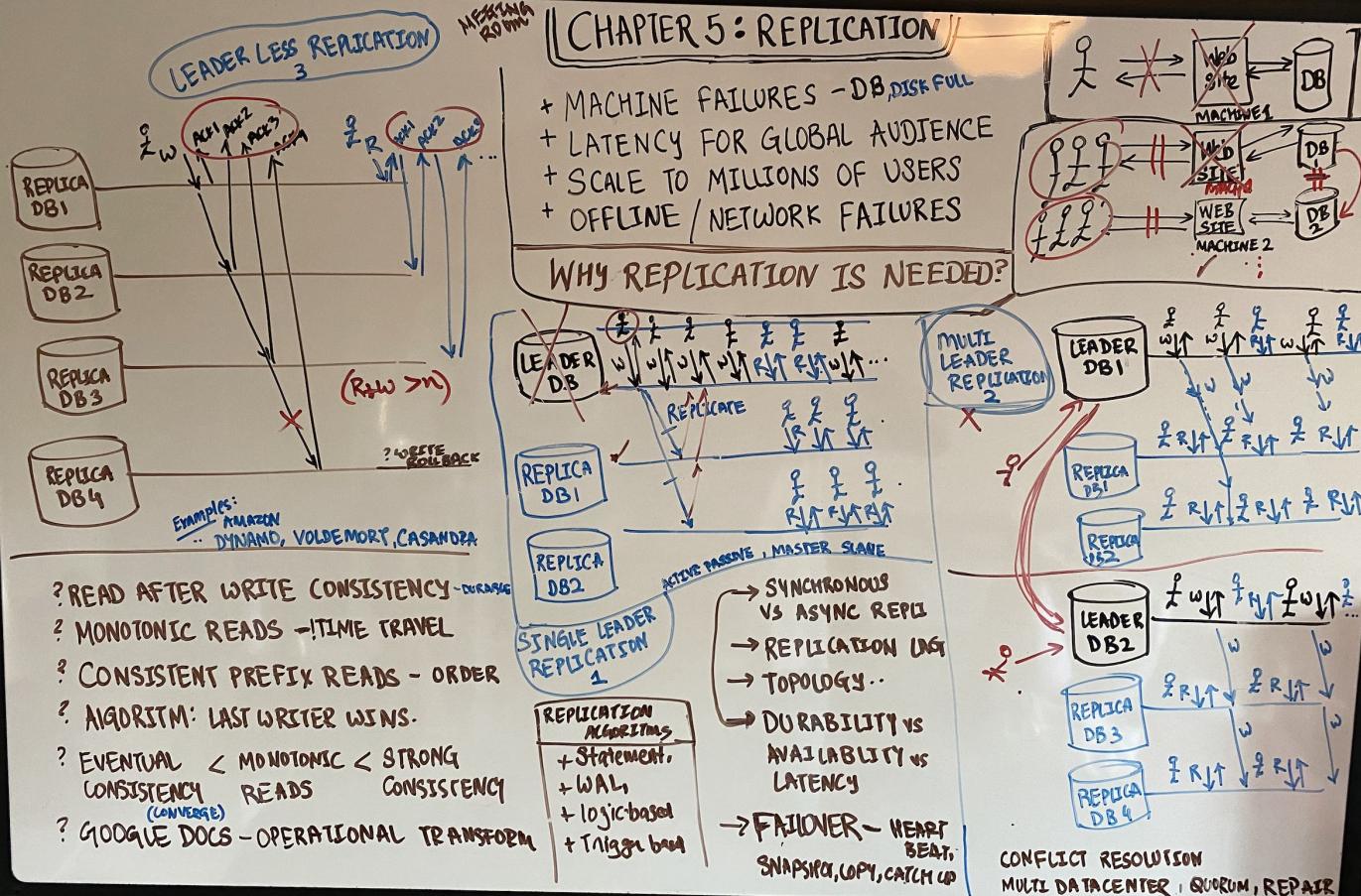
CHAPTER 5

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?



Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



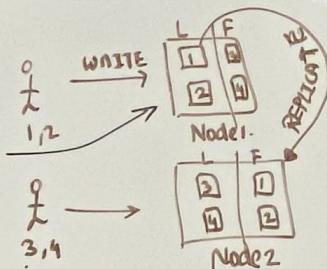
CHAPTER 6

PARTITIONING

CHAPTER-6: DATABASE PARTITIONING

+ PARTITIONING = SPLITTING = SHARDING

+ WHY? → SCALABILITY (DATA + USER + MACHINE)



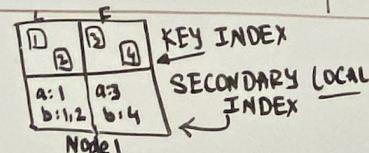
E.G.

→ 2 NODES

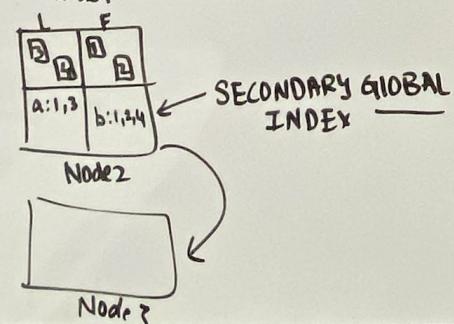
→ EACH 4 PARTITIONS

? HOT SPOTS / SKLEWS → JB

? KEY-HASH BASED PARTITION

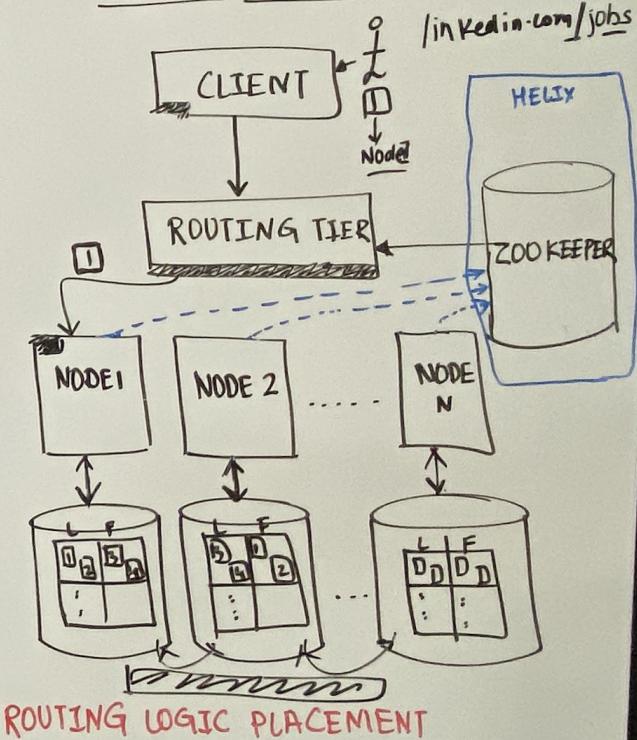


SECONDARY INDEX



? REBALANCING STRATEGIES

REQUEST ROUTING



Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



CHAPTER 7

TRANSACTIONS

CHAPTER 7 - TRANSACTIONS

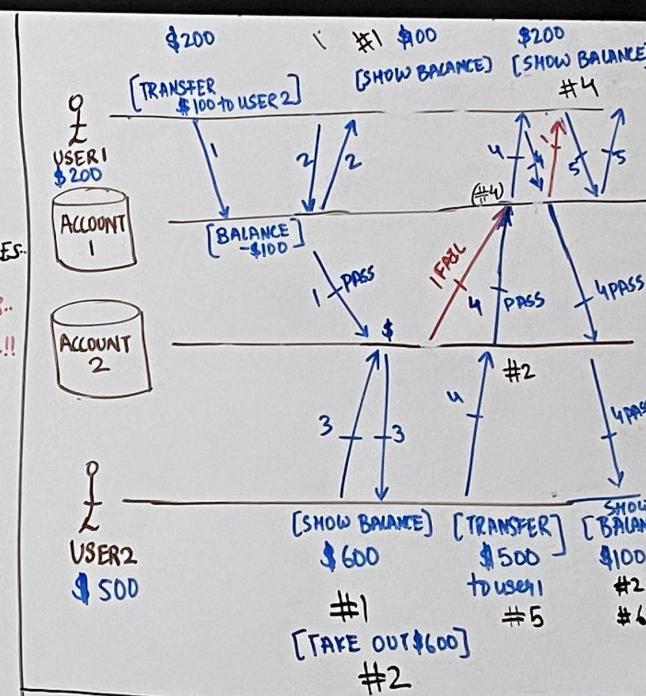
- A - ATOMICITY \Leftrightarrow All or Nothing \rightarrow ABORTABILITY
- C - CONSISTENCY \Leftrightarrow FOREIGN KEY CONSTRAINTS...
- I - ISOLATION \Leftrightarrow SERIALIZABILITY
- D - DURABILITY \Leftrightarrow STORAGE + RECOVERY GUARANTEES

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

WEAK ISOLATION LEVELS



1. NO DIRTY READS \Rightarrow READ COMMITTED
2. NO DIRTY WRITES
3. SNAPSHOT ISOLATION - REPEATED READS
4. NO LOST UPDATES - OVERWRITE
5. NO WRITE SKLEWS - PREMISE FALSE
6. NO PHANTOMS READS - \downarrow READ INVALIDATED
7. SERIALIZABILITY: a) ACTUAL SERIALIZATION
 b) TWO PHASE LOCKING
 c) SERIALIZABLE SNAPSHOT ISOLATION



$\downarrow \uparrow \downarrow \uparrow \downarrow \uparrow \rightarrow$ | CPU CORE | THREAD FOR WRITES / PARTITION

\rightarrow PESSIMISTIC LOCKING \rightarrow LOCK & WAIT

\rightarrow OPTIMISTIC LOCKING \rightarrow ABORT IF NOT SERIAL..

Designing Data-Intensive Applications

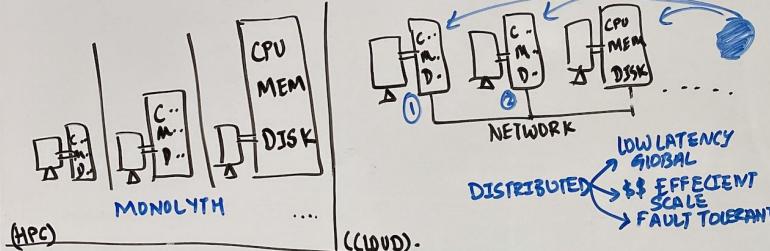
THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



CHAPTER 8

Troubles with Distributed System

CHAPTER - 8: TROUBLES WITH DISTRIBUTED SYSTEMS



- * SOFTWARE IS DETERMINISTIC!! $A \xrightarrow{\text{SOFTWARE}} B$ - ALWAYS!!
- * REAL WORLD PARTIAL FAILURES.
- * MONOLYTH: CUSTOM HARDWARE, DOWNTIME, CUSTOM NETWORK TOPOLOGY, !ROLLING UPDATES SHARED EVERYTHING, RELIABLE SLA'S ↳ NO NETWORK FAULTS - SHARKS, FIRE...

* DISTRIBUTED: 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 Mbit/s]

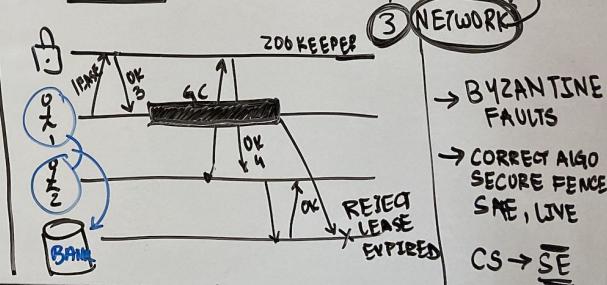
DEDICATED

UNRELIABLE CLOCKS

- QUARTZ CRYSTAL OSCILLATOR ← NTP SYNC
 - MONOTONIC CLOCK, TIME OF DAY CLOCK
System.nanoTime(), System.currentTimeMillis()
 - CLOCK DRIFTS: LEAP SECONDS
 - PRECISION TIME PROTOCOL - GPS RECEIVERS
 - SILENT & SUBLIME DATA LOSS - LRW - ORDERING
 - LOGICAL CLOCK - GOOGLE SPANNER - ORDERING RANGE (AZURE COSMOS) CONFIDENCE INT.
- PROCESS PAUSE 2 DETECT → RECOVER FAILURES

PROCESS PAUSE

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



→ BYZANTINE FAULTS

→ CORRECT ALGO SECURE FENCE SAME, 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)

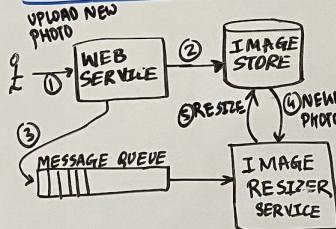
① EXAMPLES

- HUSBAND & WIFE TAKE MONEY OUT FROM SAME ACCOUNT
- NO DOCTOR ON CALL HOSPITAL
- FOOTBALL RESULTS DIFFERENCE
- OUT OF ORDER MESSAGES...
- SAME SEAT ASSIGNED TO MULTIPLE PEOPLE ON FLIGHT

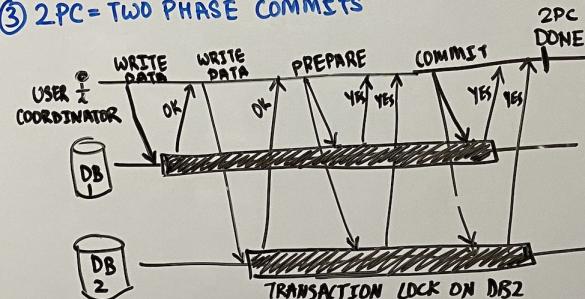
② REAL LIFE

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

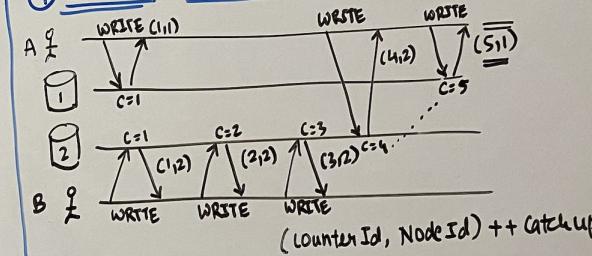
② LINEARIZABILITY



③ 2PC = TWO PHASE COMMITS



④ LAMPORT TIMESTAMP & TOTAL ORDER BROADCAST



⑤ OTHER TOPICS IN THIS CHAPTER

- + EVENTUAL CONSISTENCY == CONVERGENCE
- + NETWORK PARTITION SIMULATE
 - + STRONGER GUARANTEES
 - ↓ LOW THROUGHPUT
 - ↓ HIGH LATENCY
- + FIDELITY: READ YOUR OWN WRITES
- + SPLIT BRAIN
- + CAUSALITY << LINEARIZABLE TOTAL ORDER
- + NO OF FAILURES (10X SLOW)
 - ↓ HUMAN INTERVENTION

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



CHAPTER 10

BATCH PROCESSING

CHAPTER 10: BATCH PROCESSING

WHY?
SUMMARY/INSIGHTS/A/B TEST/BUILDING INDEX/(K,V STORE)
PYMK!!

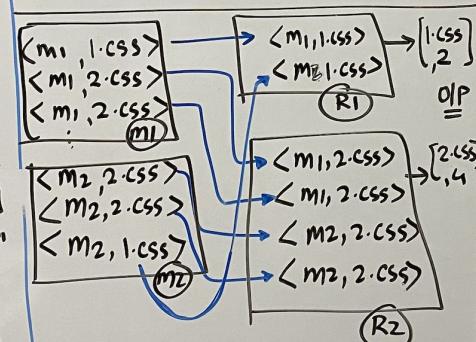
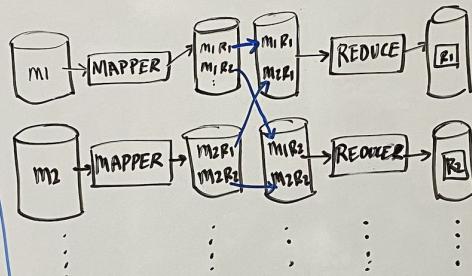
① UNIX BATCH PROCESSING

```
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/5.0..."

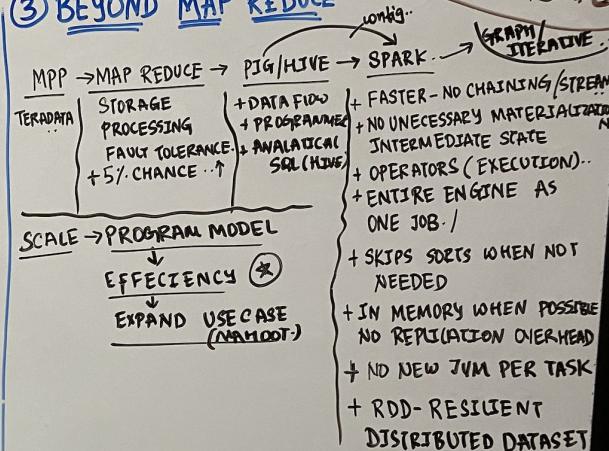
UNIFORM, LOGIC SEPARATE, EXPLICIT
 ITERATE/

② MAP REDUCE

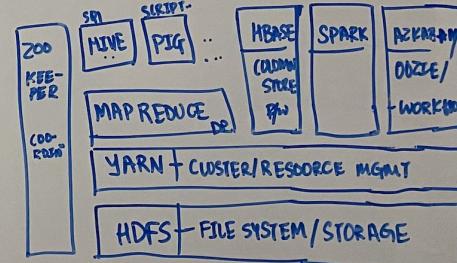


JOIN, RERUN (FAULT TOLERANT)
 AND BROADCAST HASH, PARTITION HASH.. HCATALOG
 AZKABAN, LOCALIZED COMPUTE, HOT KEYS
 10K MACHINES, PETABYTES, FAULT TOLERANT

③ BEYOND MAP REDUCE



HADOOP ECOSYSTEM: QUICK INTRO...



Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



CHAPTER 11

STREAM PROCESSING

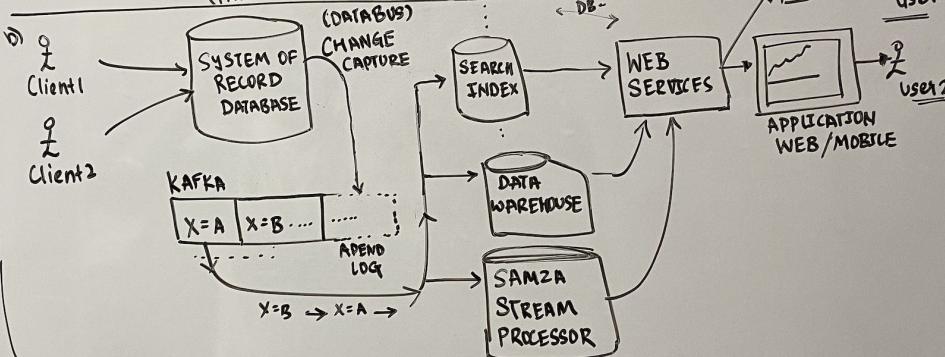
CHAPTER 11: STREAM PROCESSING

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



④ IMPORTANT CONCEPTS

* COMPARISONS: LATENCY, THROUGHPUT, ORDERING, BACKPRESSURE, ELASTICITY, STATE MANAGEMENT, STRICHTNESS DELIVERY

* POLLING VS NOTIFICATION → DB TRIGGER * BROKER RESPONSIBLE FOR DURABILITY / RETENTION (DATA)

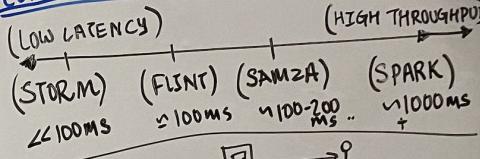
* STREAMING ALLOWS FOR MULTIPLE VIEWS OF THE SAME DATA → N/B TEST → NO SCHEMA EVOLUTION....

* 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..

③ COMPARISONS



Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE 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)

CHAPTER-12 FUTURE OF DATA SYSTEMS

*) 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.

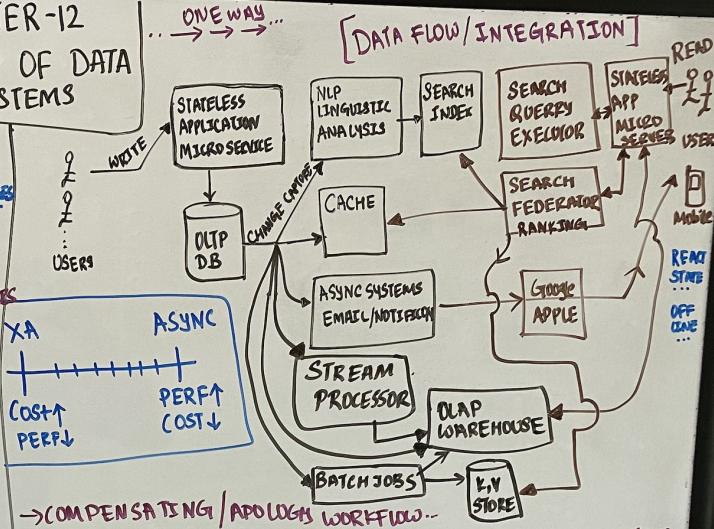
*) CAUSAL RELATIONSHIPS: Email sends, GROUP SUB,
 CONSTRAINTS (BANK BALANCE)
 (FOCUS 1 THING)

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

*) ORDERED LOG EVENTS + IDEMPOTENT CONSUMER: ^{vs}
 ↳ END TO END REQUEST ID, DE DUPE AT CONSUMER..

→ (R) - DEBIT \$10 FROM A, CREDIT \$10 TO B - LOG
 (R, D) → DEBIT \$10, (R, C) → CREDIT \$10 → CRASH / REPAIR
 ↓
 DONE DONT REDO .. NOT DONE → FINISH..] CONSUMER
 dedupe(R..)

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



DO THE RIGHT THING

→ PREDICTIVE ANALYTICS MERELY CODIFY THE PAST... / PEOPLE / ..
 → BIAS / ECHO CHAMBERS / SYSTEMS THINKING - UNINTENDED CONSEQUENCES...

PRIVACY & TRACKING! = SURVEILLANCE...
 (ENV) (POLITICS)



Introduction to CDN