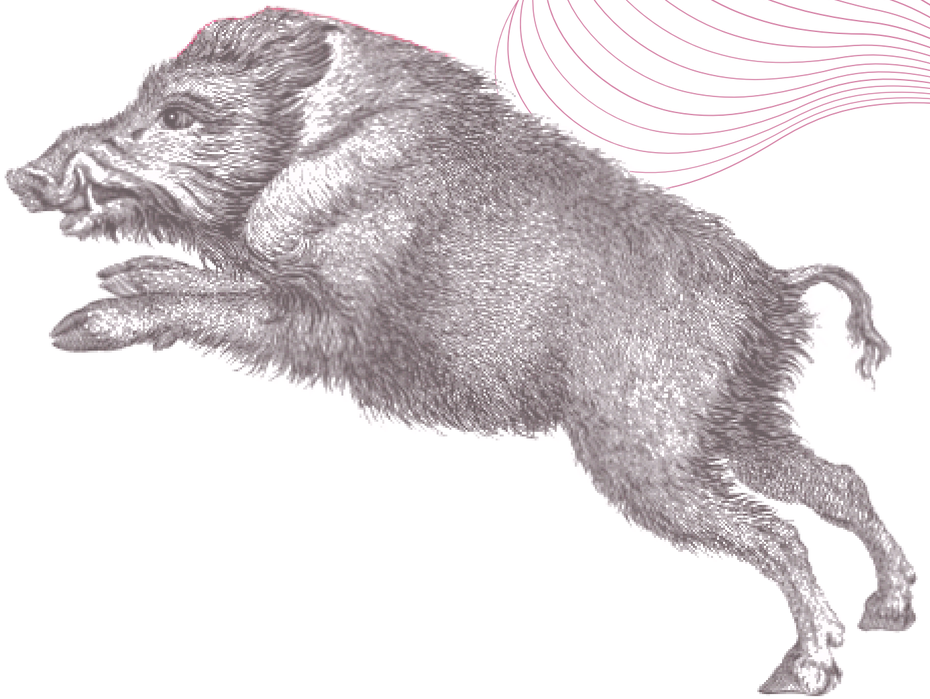
Designing Data-Intensive Applications



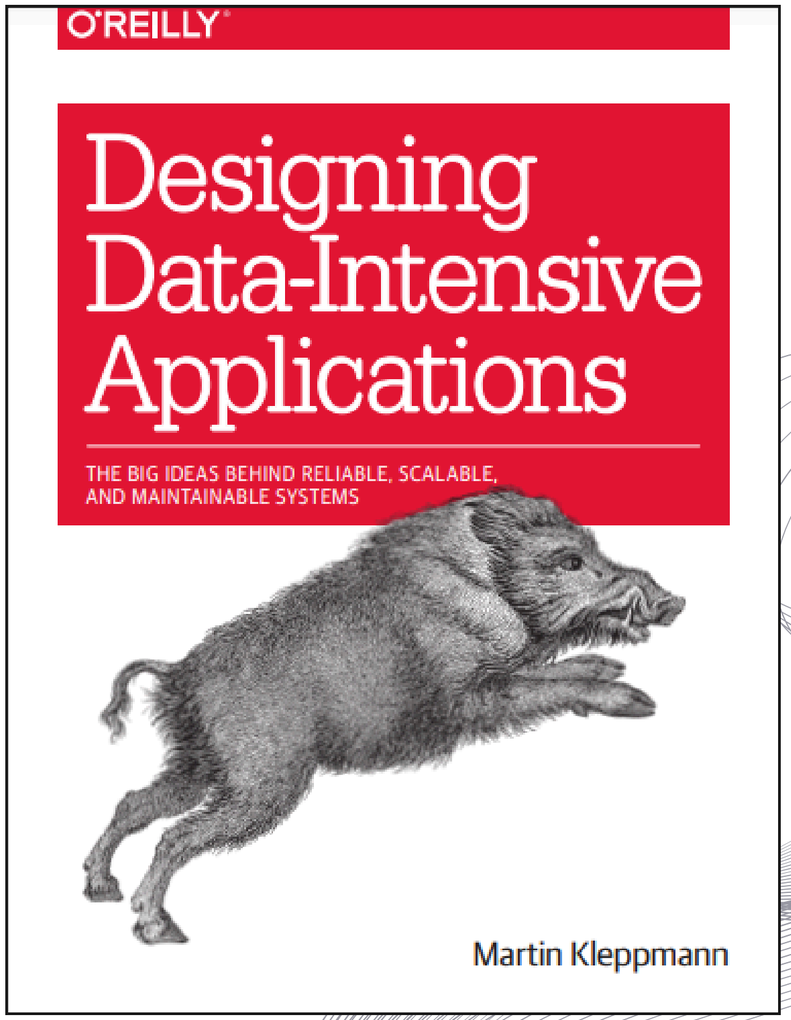
A book by:
Martin Kleppmann

Martin Kleppman

- Professor at the University of Cambridge
- Published Designing Data-Intensive Applications 2017.
- Martin Kleppman co-founded Rapportive (acquired by LinkedIn in 2012) and Go Test It (acquired by Red Gate Software in 2009)

Book Introduction

- Data intensive >>>>>>> Compute Intensive
- Data systems are such a successful abstraction that we use them all of the time without thinking about it
- This book will talk about the principles and practicalities of data systems.



Data Intensive App Example

Why It's Data-Intensive?

- **Massive Data Volume**: Processes millions of taxpayer records, transactions, and documents daily.
- **Complex Processes:** Validates ever-changing tax rules with intricate data relationships.
- **Distributed System Complexity:** Relies on many interconnected services (e.g., banks, regional tax offices, external services) that must stay continuously synchronized.

What happen?

- **Service Disruption**: Frequent system crashes
- **Hybrid System Rollback**: Due to instability, they reverted to a parallel system with the old tax platform.
- **Financial & Public Trust Loss:** Despite a Rp 1.3 trillion budget, issues persist, leading to extra costs and eroding public confidence.

Book's Part

Foundation

1. Reliable, Scalable, and Maintainable Applications
2. Data Models and Query Languages
3. Storage and Retrieval
4. Encoding and Evolution

Distributed Data

5. Replication
6. Partitioning
7. Transactions
8. The Trouble with Distributed Systems
9. Consistency and Consensus

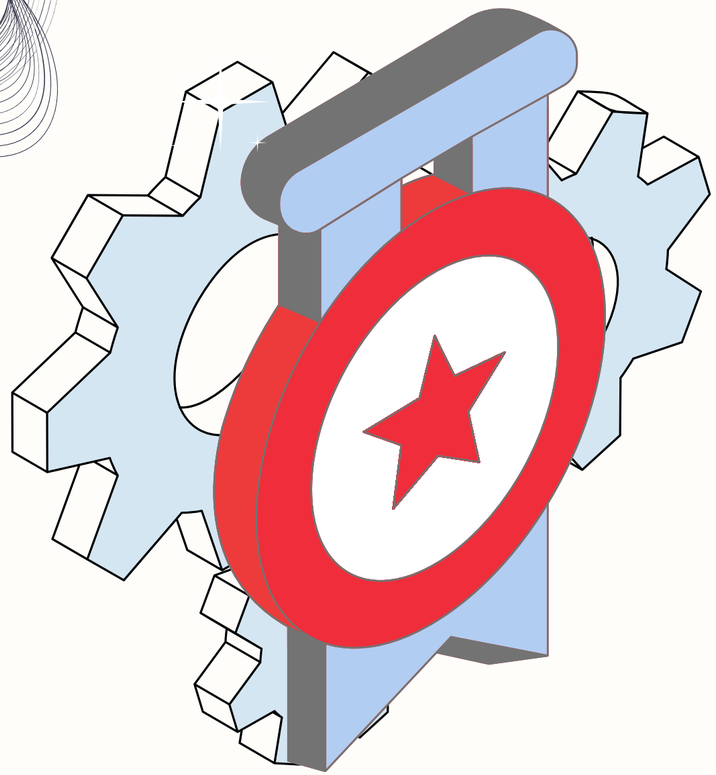
Derived Data

10. Batch Processing
11. Stream Processing.
12. The Future of Data Systems

Chapter 1

Reliable, Scalable, and Maintainable Applications

Reliability



Definition

Continuing to work correctly even when things go wrong

Why???

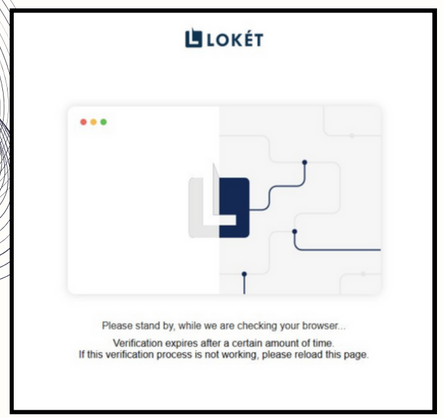
To find a solution or an alternative when something goes wrong

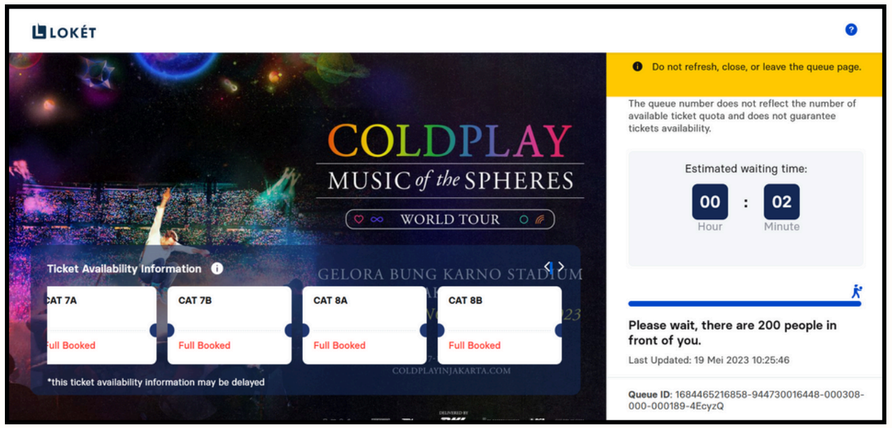
Reliability

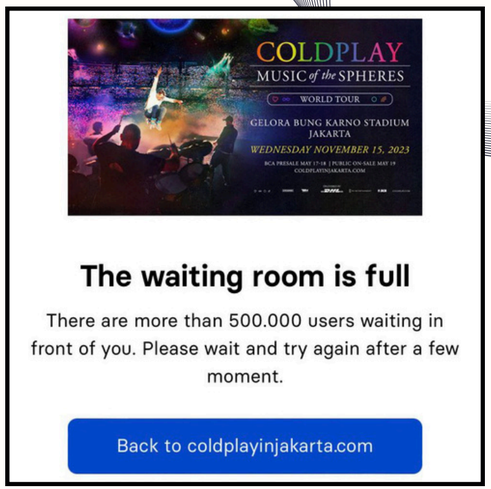
Typical Expectation

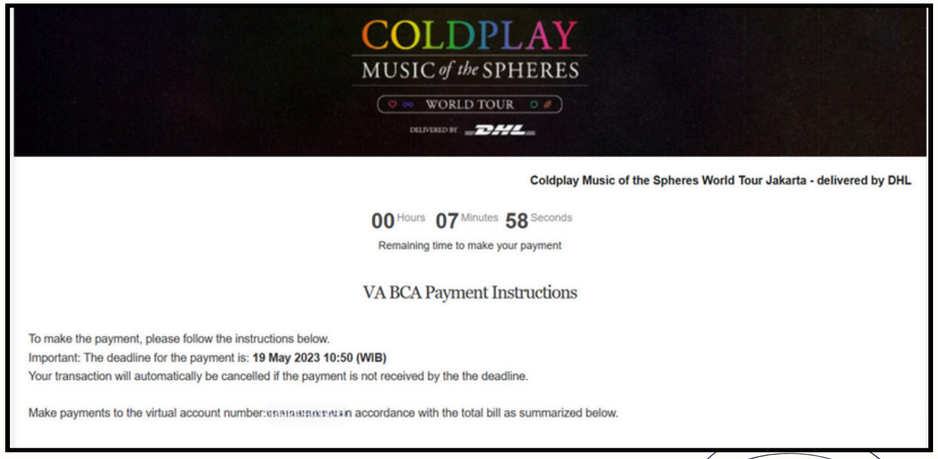
- Application performs the function the user expected
- Tolerate the user making mistakes or using the software in unexpected way
- Its performance is good
- The system prevents unauthorized access and abuse.

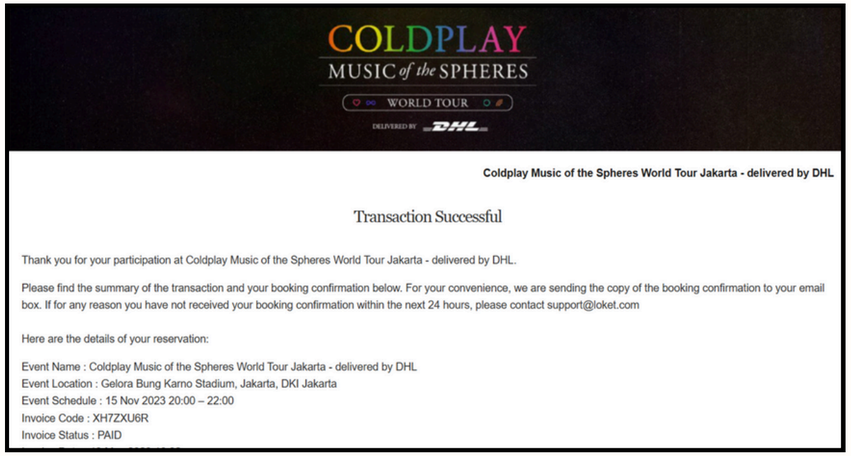
Examples of Good Reliability











Kinds of Faults

Hardware Faults

Software Faults

Human Errors

Hardware Faults

Physical failures in hardware that can disrupt the system. We can improve resilience to hardware failures by using hardware redundancy (RAID, dual power supplies, hot-swappable components)

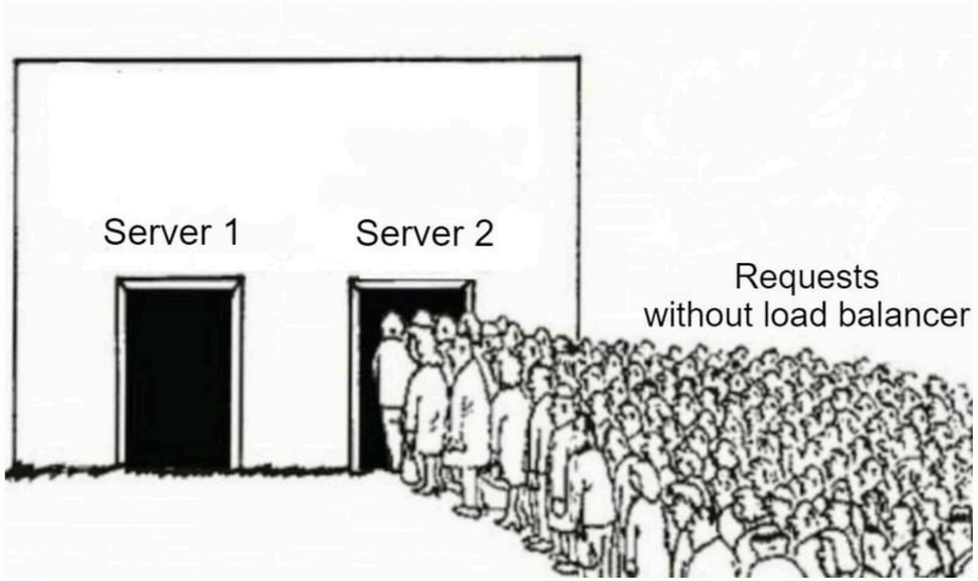
Example with hardware redudancy

- **Initial State:** Server has 2 PSUs, one connected to the main power, the other to a UPS/generator.
- **Issue Occurs:** PSU 1 fails, but the server keeps running on PSU 2
- **Action Taken:** Team gets a notification and replaces PSU 1 without shutting down the server (hot-swap).
- **Recovery:** New PSU is installed, and the server returns to using both power sources.
- **Result:** No downtime, data remains safe, and services

Hardware Faults

Software fault tolerance techniques

- Modern systems adopt software fault-tolerance techniques (data replication, load balancing, rolling upgrades)



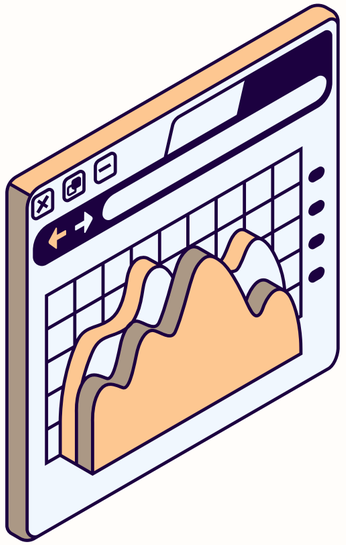
Example

- **Initial State:** The app runs on 3 servers with a load balancer.
- **Issue:** 1 server crashes, but the other 2 handle requests.
- **Action:** The load balancer redirects traffic to the active servers.
- **Recovery:** After repair, the server is reintegrated automatically.
- **Result:** Zero downtime, uninterrupted service, and system stability.

Software Faults

Characteristic

- Difficult to detect
- Systematic & cascading failures
- No quick fix



Example

- **Initial State:** A video streaming app with multiple microservices.
- **Issue:** A memory leak in the recommendation service spikes CPU and RAM, disrupting playback.
- **Action:** Engineers isolate, fix, test, and redeploy the service.
- **Recovery:** The service is gradually restored with performance monitoring.
- **Result:** Core services stay online, downtime is minimized, and the bug is resolved.

Tolerating Faults

Human Errors

Causes

- Misconfiguration settings
- Lack of Understanding
- Pressure & Fatigue

Solution

- Minimize Opportunities for Mistakes
- Decouple High-Risk Actions
- Enable Quick & Easy Recovery
- Use Detailed & Clear Monitoring
- Provide Good Management & Training

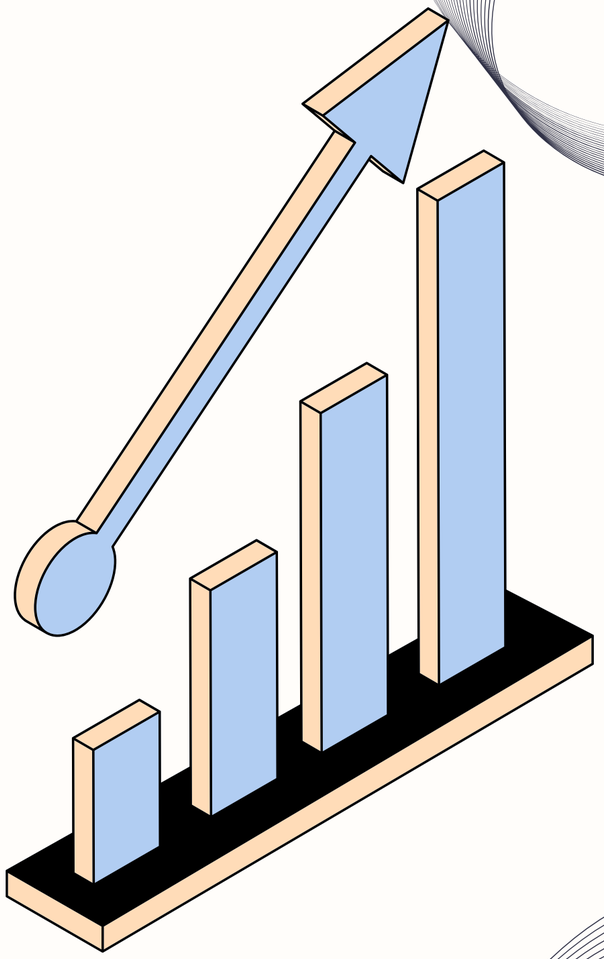
Scalability

Definition

Asking if the system grows in a particular way, what are the options for coping with that growth

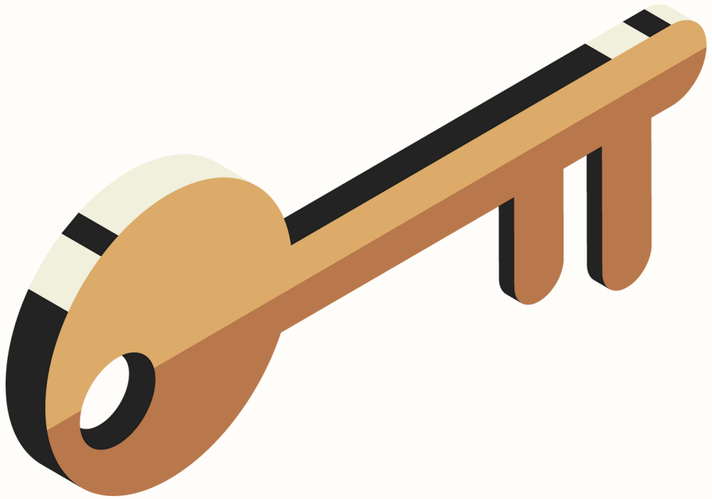
Why???

A system should be able to handle additional loads, such as an increasing number of users or data volume, without experiencing a decrease in performance



Scalability

Key Considerations



- Load
- Perfomance
- How to cope with load?

Scalability

Load

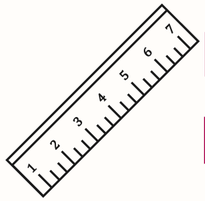
- Requests per second
- Read/Write ratio
- Active users
- Cache hit rate

Scalability

Perfomance

Latency

Response time



Perfomance measured by percentiles

Percentiles are used in SLA

Scalability

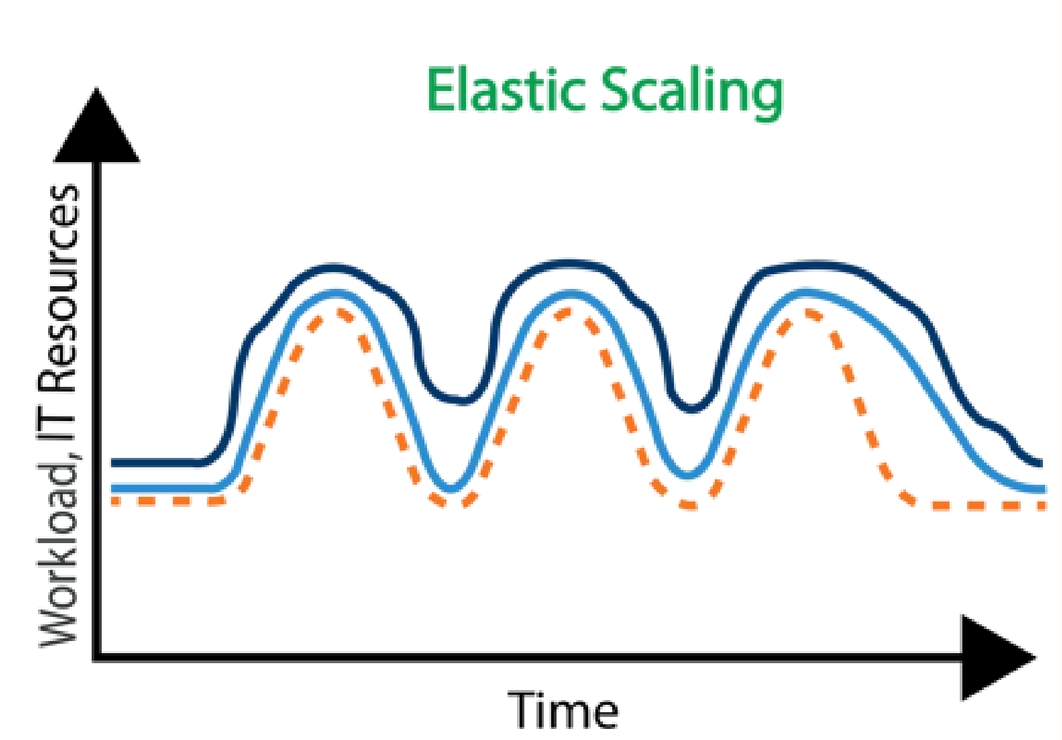
How to Cope with Load?

Scale Up

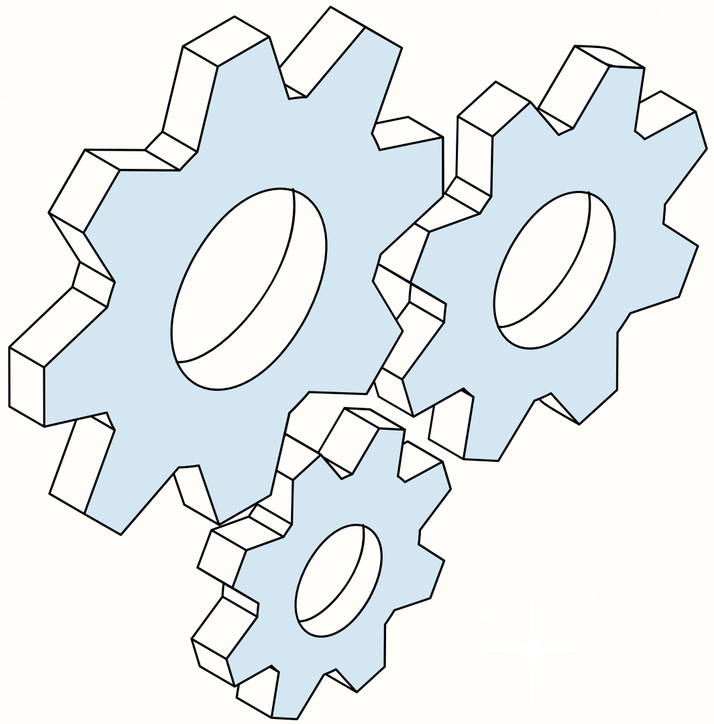
Scale Out

Scalability

How to Cope with Load?



Maintainability



Definition

Keeping the system easy to modify and adaptable for future changes or improvements

Why???

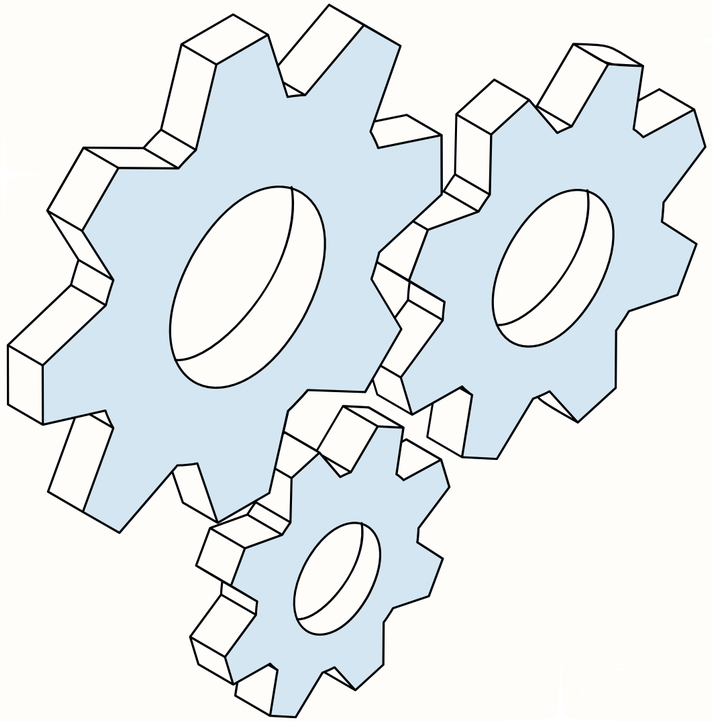
Eases the burden on engineers who work with the system by preventing difficulties in debugging, developing new features, and fixing bugs, which can lead to higher operational costs

Maintainability

Lack of Maintainability Example

- Initial State: An e-commerce app lacks clear documentation and organized code.
- Issue: Adding a seasonal discount feature is difficult due to overlapping functions and inconsistent variables.
- Impact: Slower development, more bugs, and longer debugging times.
- Result: Delayed feature release, lost revenue, and higher costs to fix the code.

Maintainability



- Operability
- Simplicity
- Evolvability

Operability

Measuring how easy it is for the IT team to operate and maintain the system on a daily basis

How?

- Monitoring system and quickly restoring
- Tracking system failure and degraded perfomance
- Keeping software up to date
- Keeping track how one system affect the other
- Establishing good deployment, configurations practices
- Avoiding dependency or individual machines
- Good documentation

Simplicity

What is simplicity?

- The simpler the design and code, the easier it is engineers to understand

How to reduce complexity

- Clean Code
- Modular Design
- Use descriptive and consistent names
- Refactoring
- Using design patterns
- Documentation & comments

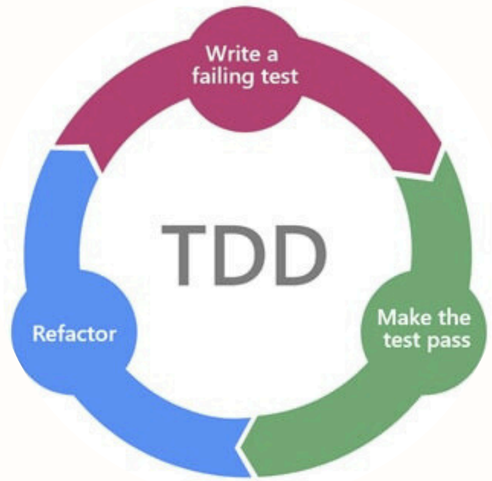
Evolvability

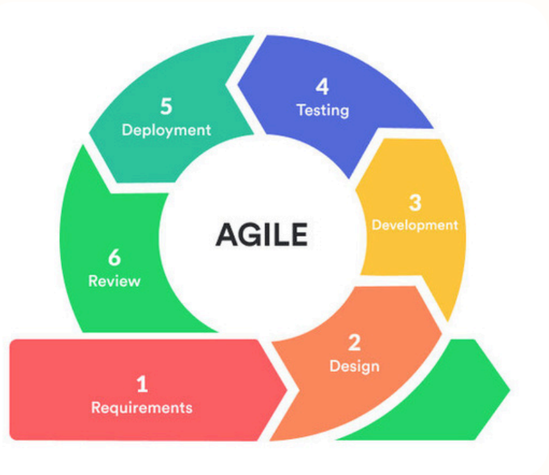
Making it easy for engineers to make changes into the system in the failure

Why it is needed?

- Business Priorities change
- Users request new features
- New plaforms
- Architectural changes

How?





Chapter 2

Data Models and Query Languages

Data Models

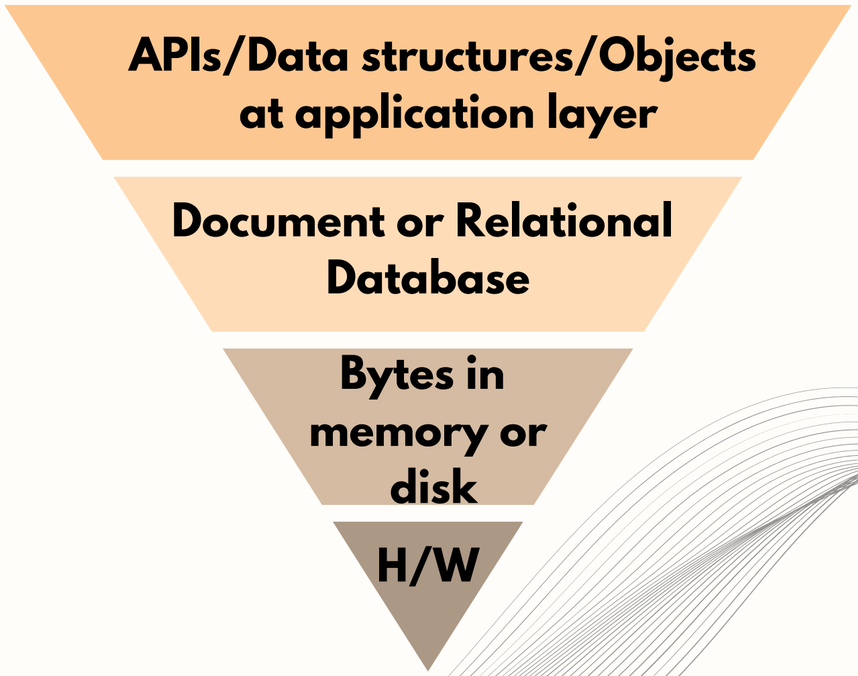
Definition

A way to represent data at a level that is easier for humans and systems to understand

Importance

Data models shape how we think about problems, influence how software is designed, and determine which operations are efficient or challenging

Abstraction



Data Models

Relational

Document

Graph

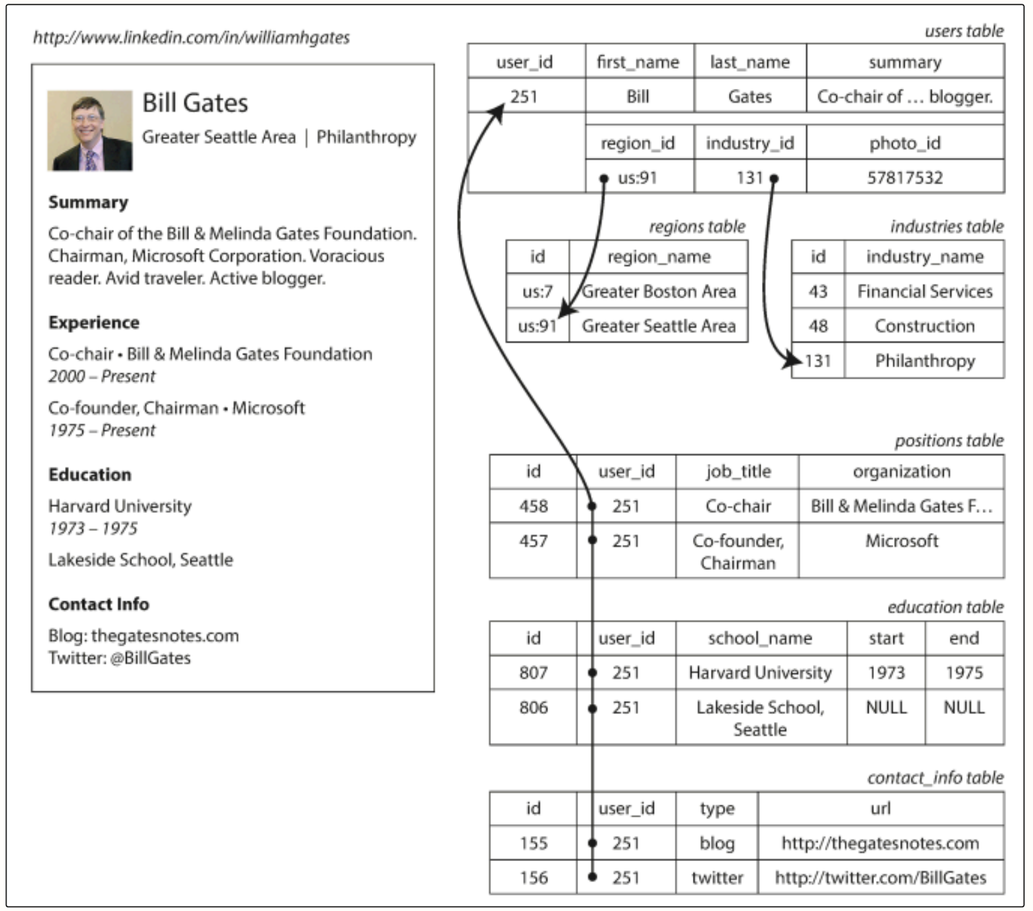
Relational

Definition

- A relational database organizes data into relations, also known as tables, where each relation is an unordered collection of tuples (rows).
- Schema on write

Problem

- Impedance mismatch
- Limited Scalability
- Rigid Structure
- Excessive Joins



http://www.linkedin.com/in/williamhgates

Bill Gates

Greater Seattle Area | Philanthropy

Summary

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

Experience

Co-chair • Bill & Melinda Gates Foundation
2000 – Present

Co-founder, Chairman • Microsoft
1975 – Present

Education

Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

users table

user_id	first_name	last_name	summary
251	Bill	Gates	Co-chair of … blogger.

region_id	industry_id	photo_id
us:91	131	57817532

regions table

id	region_name
us:7	Greater Boston Area
us:91	Greater Seattle Area

industries table

id	industry_name
43	Financial Services
48	Construction
131	Philanthropy

positions table

id	user_id	job_title	organization
458	251	Co-chair	Bill & Melinda Gates F…
457	251	Co-founder, Chairman	Microsoft

education table

id	user_id	school_name	start	end
807	251	Harvard University	1973	1975
806	251	Lakeside School, Seattle	NULL	NULL

contact_info table

id	user_id	type	url
155	251	blog	http://thegatesnotes.com
156	251	twitter	http://twitter.com/BillGates

Document

NoSQL

- A type of NoSQL database that stores, retrieves, and manages data in document format, typically JSON, BSON, or XML.
- Schema on Read


```
{
  "user_id":     251,
  "first_name":  "Bill",
  "last_name":   "Gates",
  "summary":     "Co-chair of the Bill & Melinda Gates... Active blogger.",
  "region_id":   "us:91",
  "industry_id": 131,
  "photo_url":   "/p/7/000/253/05b/308dd6e.jpg",
  "positions": [
    {"job_title": "Co-chair", "organization": "Bill & Melinda Gates Foundation"},
    {"job_title": "Co-founder, Chairman", "organization": "Microsoft"}
  ],
  "education": [
    {"school_name": "Harvard University",       "start": 1973, "end": 1975},
    {"school_name": "Lakeside School, Seattle", "start": null, "end": null}
  ],
  "contact_info": {
    "blog":    "http://thegatesnotes.com",
    "twitter": "http://twitter.com/BillGates"
  }
}
```


Problem

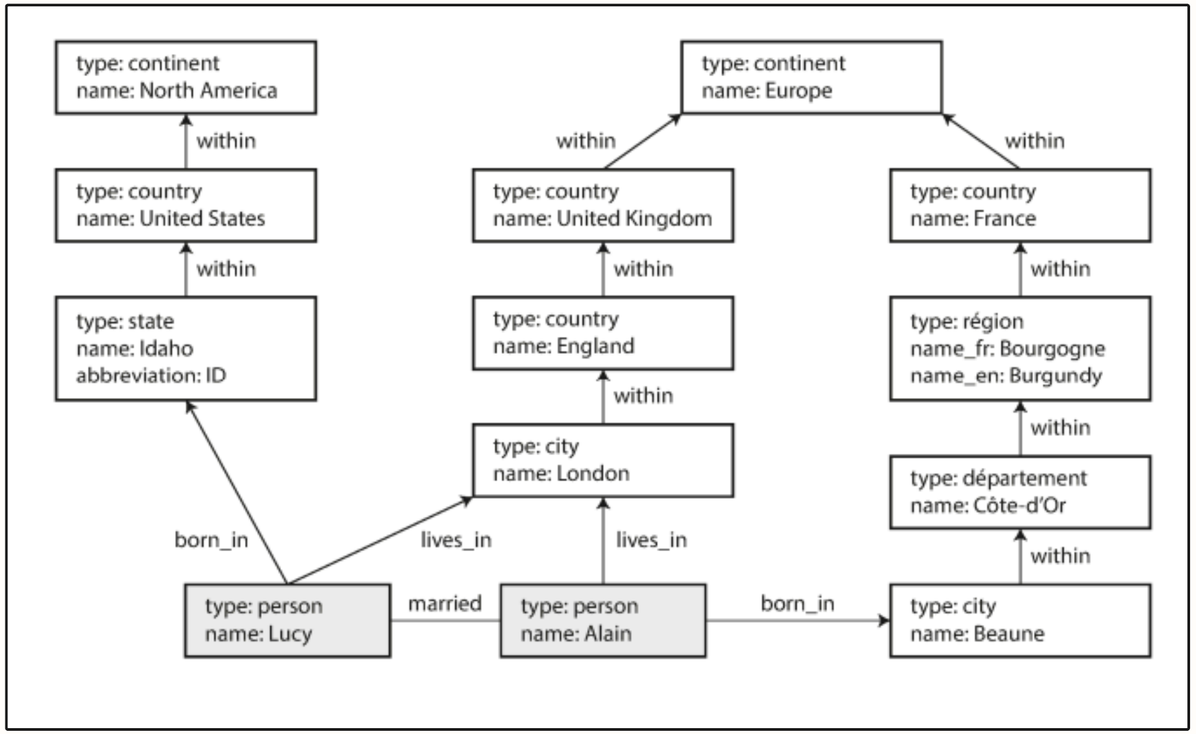
- Lack of ACID Transactions.
- Limited Querying Capabilities:
- Eventual Consistency
- Data Duplication

Graph

- A graph database represents data as nodes (entities) and edges (relationships), making it ideal for handling complex, highly connected data.

Characteristic

- Relationship-Centric
- Fast Traversal
- Flexible Schema
- Rich Metadata



Comparison

Relational	Document	Graph
Better joins	Schema flexibility	Best for complex, interconnected relationships
Better many to one and many to many relationship	Better perfomance due to locality	Efficient traversal of complex relationships
Schema on write	Schema on read	Flexible schema
Requires rewriting the entire row	Updates a document without changing the entire content	Updates and nodes independently
Reads only the necessary data	Requires reading the entire document for retrieval	Quick access to related data through graph traversals
Performance may degrade with complex queries	Performance may degrade with complex queries	Performance remains stable for complex relationship queries

Query Languages

Declarative

```
Select * from animals where family='sharks';
```

Imperative

```
function getSharks() {
    var sharks = [];
    for(var i=0;i<animals.length;i++){
        if(animals[i].family === "Sharks"){
            sharks.push(animals[i]);
        }
    }
    return sharks;
}
```

Query Languages

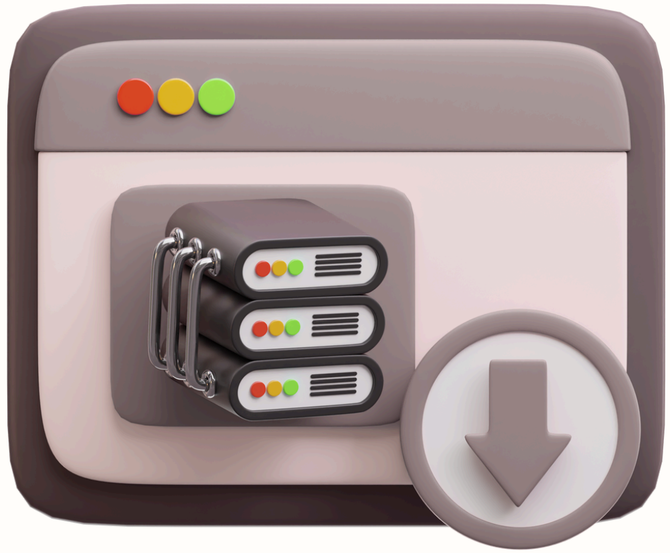
Declarative

- Specify the desired data pattern (like conditions or transformations).
- The query optimizer finds the best way to retrieve matching data.
- Easier to parallelize, as execution can be handled automatically.

Imperative

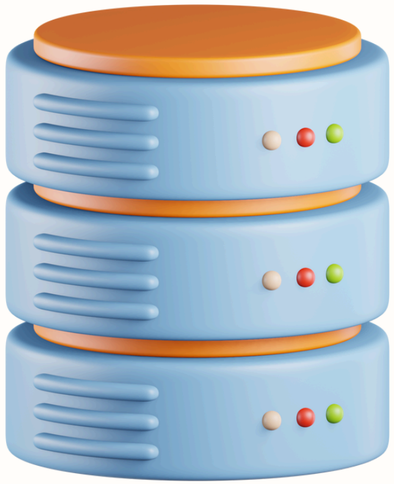
- Much more verbose but flexible. Coding the specific operations
- Parallel code harder to write for each query
- Tells the computer to perform certain operation in a certain order.
- Hard to parallelize because it specifies instructions to be executed in a particular orde

Chapter 3



Storage and Retrieval

What does a database do?



Store data

Give data when requested

As a Developer it is essential for selecting the right storage engine for your application's workload, optimizing performance, and preventing bottlenecks as usage scales.

Data Structures

Log Structure

- SS Tables
- LSM Trees

Index Based

- B-Trees
- Hash

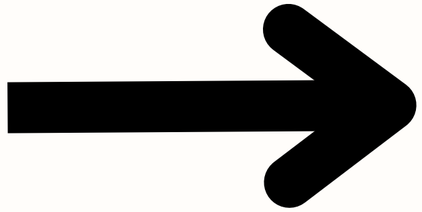
Why Index?

```
$ db_set 42 '{"name":"San Francisco","attractions":["Exploratorium"]}'

$ db_get 42
{"name":"San Francisco","attractions":["Exploratorium"]}

$ cat database
123456,{"name":"London","attractions":["Big Ben","London Eye"]}
42,{"name":"San Francisco","attractions":["Golden Gate Bridge"]}
42,{"name":"San Francisco","attractions":["Exploratorium"]}
```

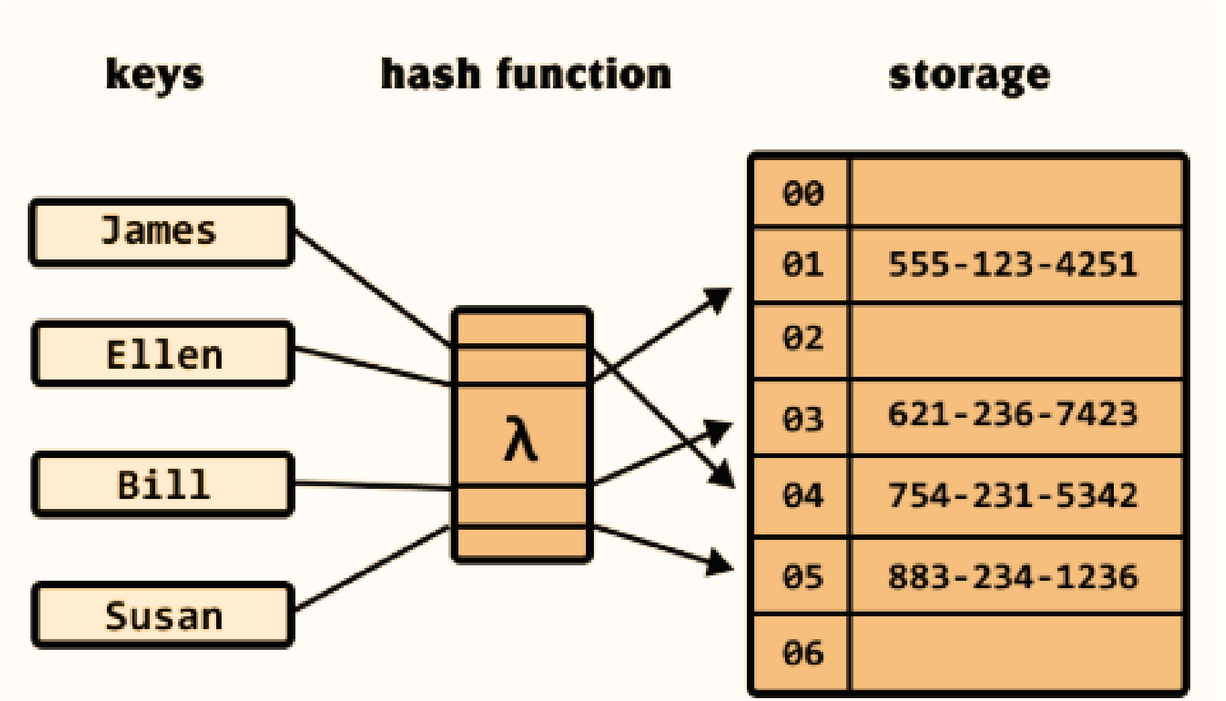
- Inefficient Lookups (O(n))
- Redundant Data Growth



- Optimized Search and Updates with Indexing

Index Based

Hash



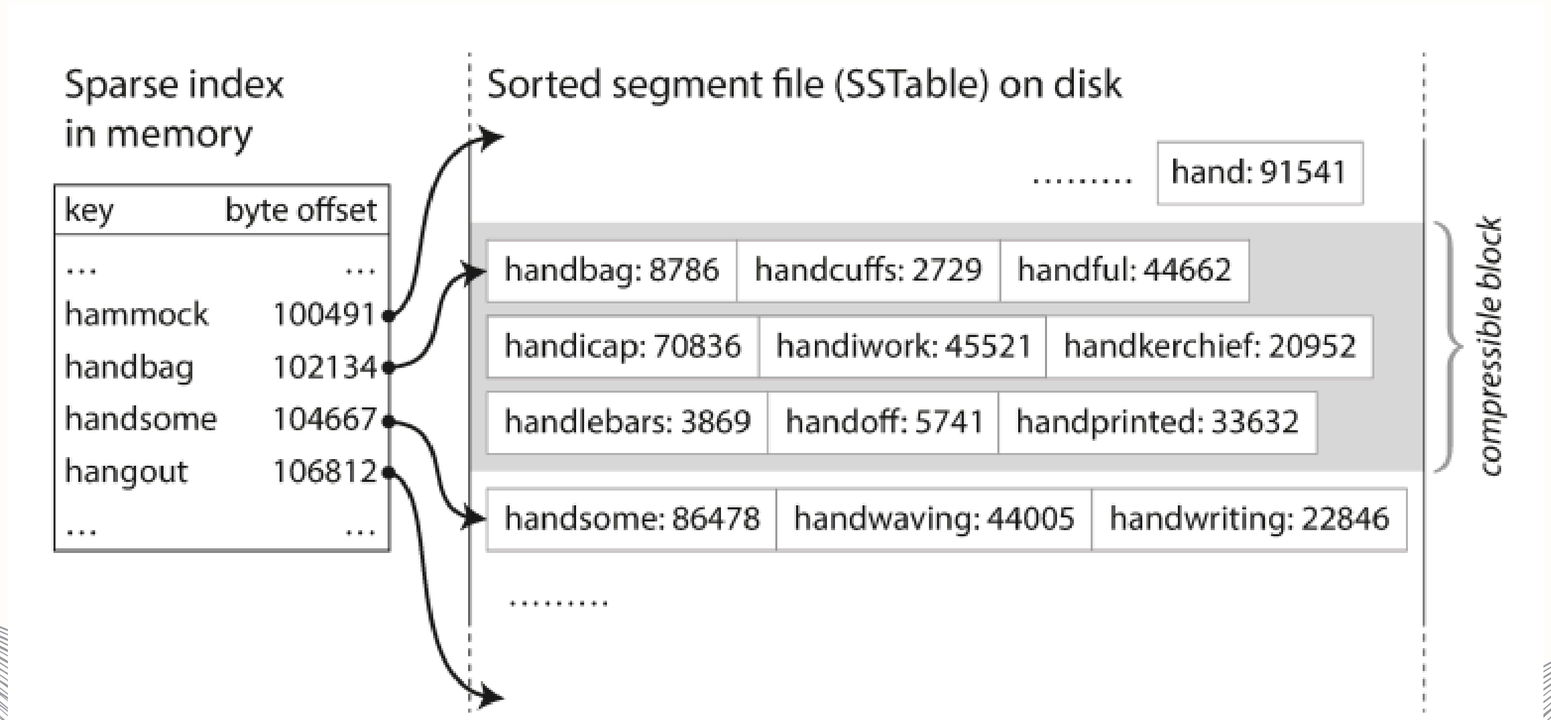
Hash Map

Limitation

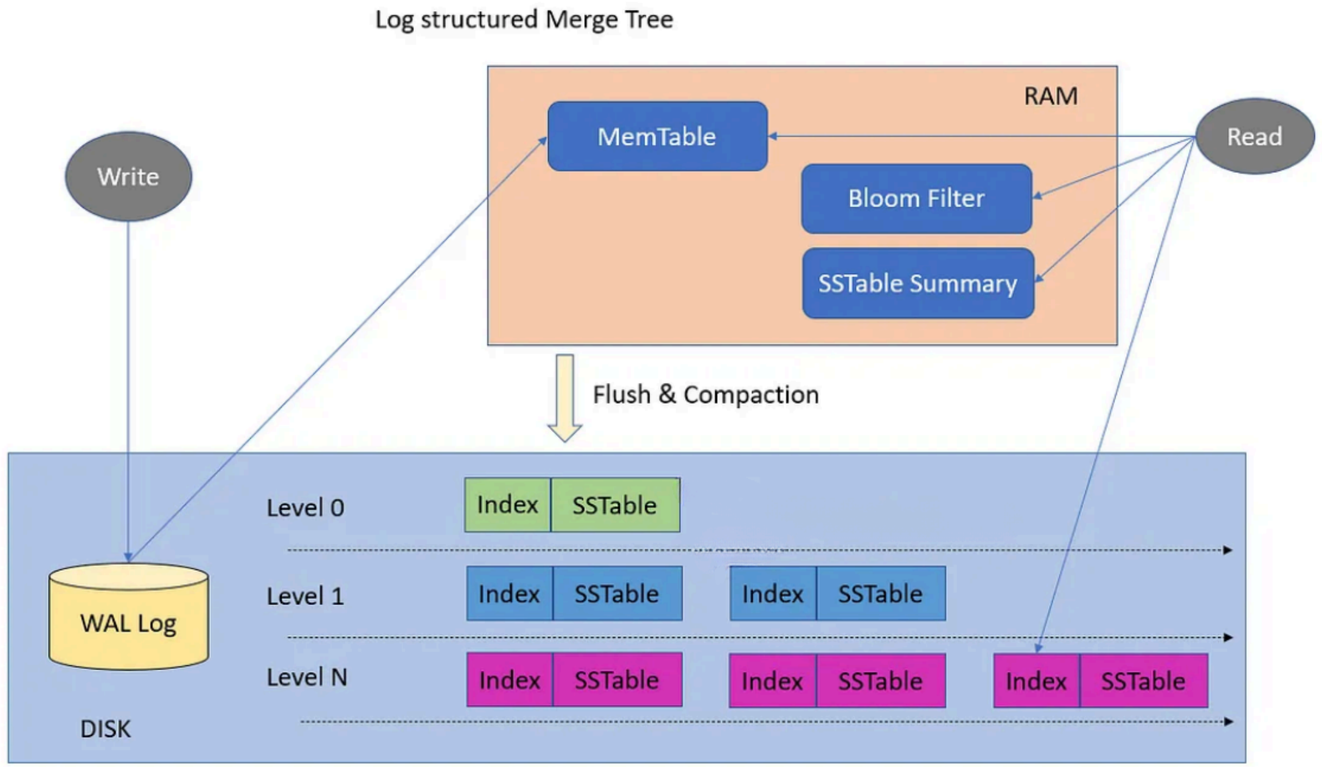
- Memory and Disk Access Limitation – The hash table must fit entirely in RAM, and if it exceeds available memory, storing it on disk leads to excessive random access, significantly slowing down operations.
- Inefficient for Range Queries – Hash tables do not support range-based lookups, requiring individual key lookups for queries within a specific range
- Collisions - that can be handled with chaining or open addressing

Data Structures

SS Tables



LSM Trees

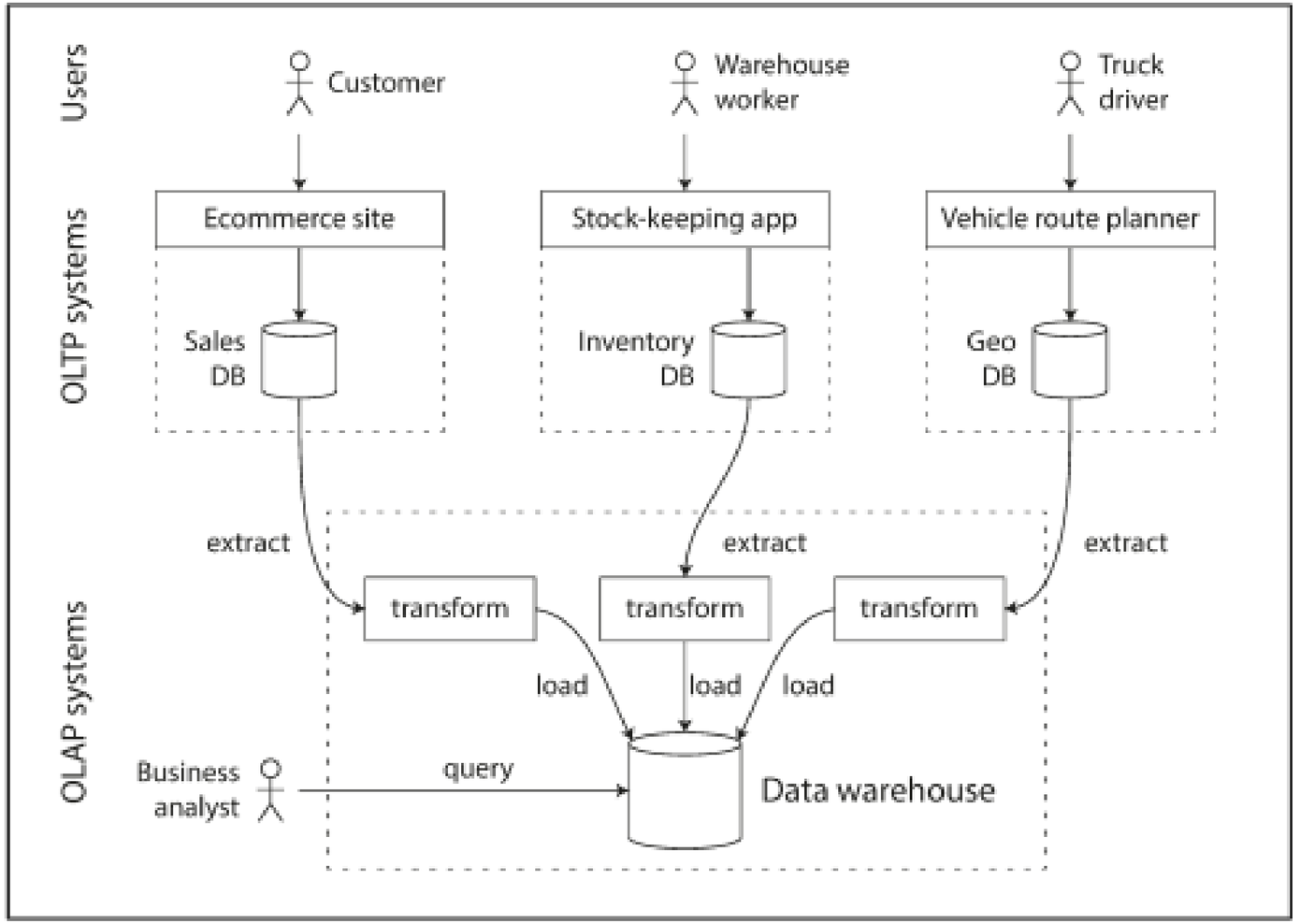

Log structured Merge Tree
RAM
MemTable
Read
Write
Bloom Filter
SSTable Summary
Flush & Compaction
Level 0
Index
SSTable
Level 1
Index
SSTable
Index
SSTable
WAL Log
Level N
Index
SSTable
Index
SSTable
Index
SSTable
DISK

Understanding data structures helps us design efficient databases, but how data is accessed depends on whether we focus on transactions or analytics.

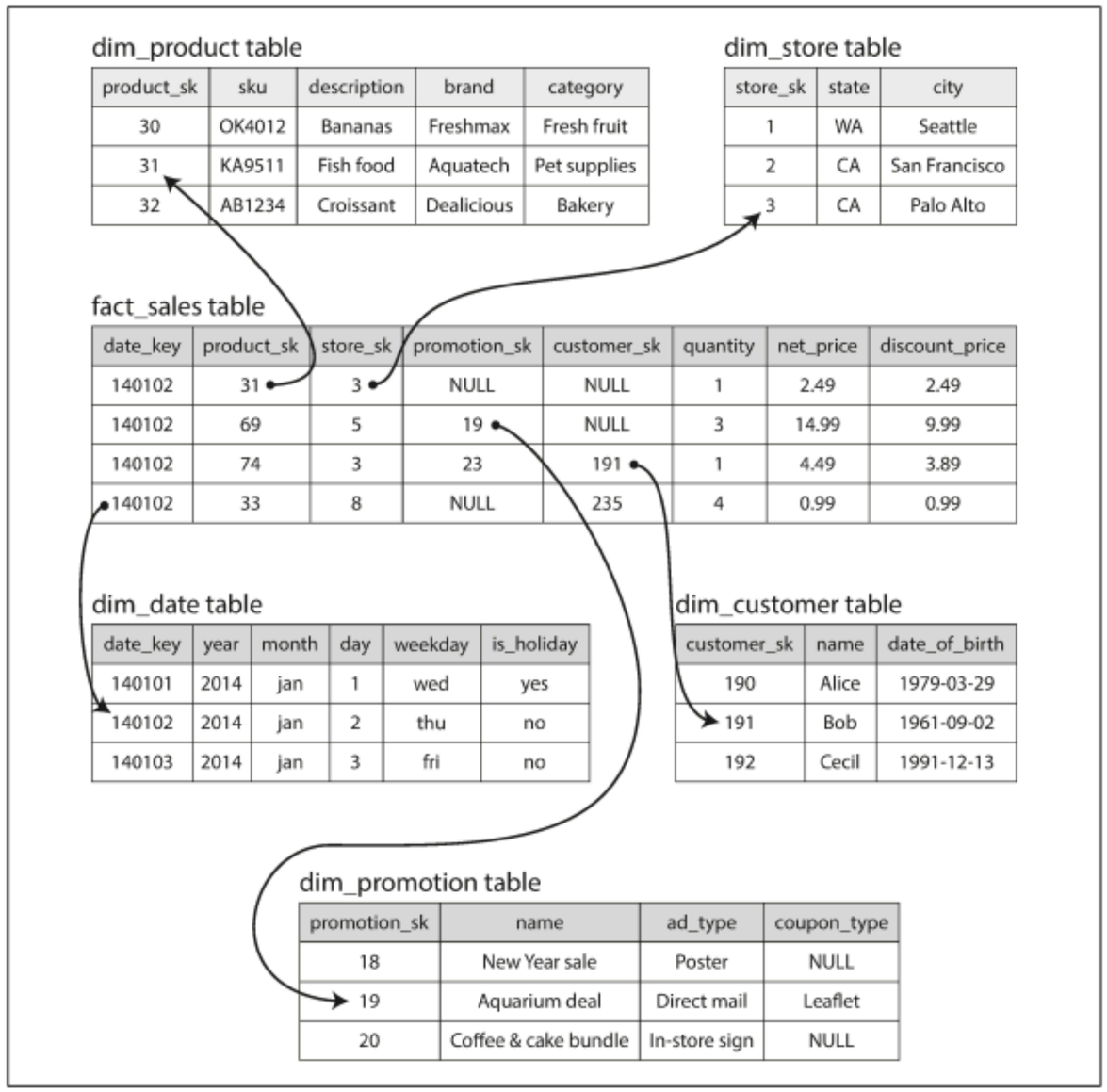
OLTP vs OLAP

Property	OLTP	OLAP
Main read pattern	Small number of reads per query, fetched by key.	Aggregate over large number of records.
Main write pattern	Random-access, low-latency writes from user input.	Bulk import (ETL) or event stream
Primarily used by	End user/customer, via web application	Internal analyst, for decision support.
What data represents	Latest state of data.	History of events that happened over time.
Dataset Size	Gigabytes to terabytes.	Terabytes to petabytes.

Extract, Transform Load(ETL) to Data Warehouse



Star Scheme in Data Warehouse



dim_product table

product_sk	sku	description	brand	category
30	OK4012	Bananas	Freshmax	Fresh fruit
31	KA9511	Fish food	Aquatech	Pet supplies
32	AB1234	Croissant	Dealicious	Bakery

dim_store table

store_sk	state	city
1	WA	Seattle
2	CA	San Francisco
3	CA	Palo Alto

fact_sales table

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	31	3	NULL	NULL	1	2.49	2.49
140102	69	5	19	NULL	3	14.99	9.99
140102	74	3	23	191	1	4.49	3.89
140102	33	8	NULL	235	4	0.99	0.99

dim_date table

date_key	year	month	day	weekday	is_holiday
140101	2014	jan	1	wed	yes
140102	2014	jan	2	thu	no
140103	2014	jan	3	fri	no

dim_customer table

customer_sk	name	date_of_birth
190	Alice	1979-03-29
191	Bob	1961-09-02
192	Cecil	1991-12-13

dim_promotion table

promotion_sk	name	ad_type	coupon_type
18	New Year sale	Poster	NULL
19	Aquarium deal	Direct mail	Leaflet
20	Coffee & cake bundle	In-store sign	NULL

Column Oriented Storage

What?

It is a method of storing data where all the values in a single column are stored together, instead of storing entire rows together.

Why?

- Row storage is fast for CRUD but inefficient for OLAP.
- Column storage reads only needed columns, reducing I/O and speeding up queries.

What do you think about Chapter 1,2,3?

Thank You So Much