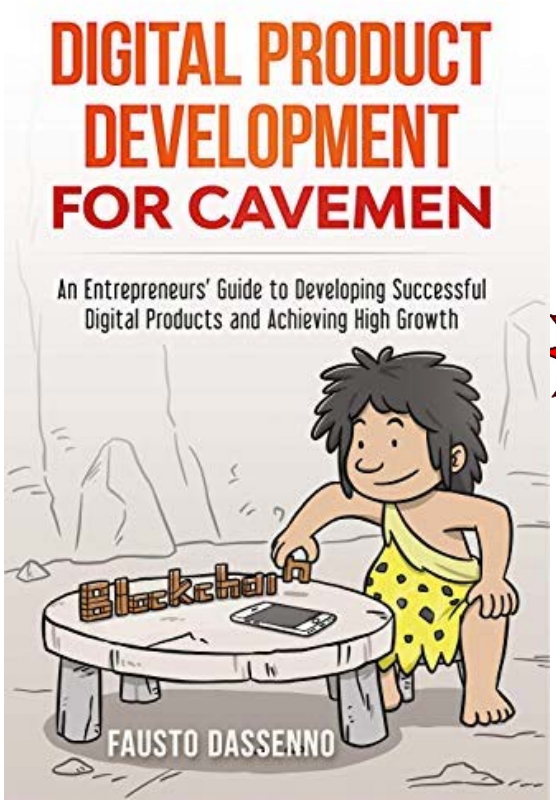
Database Systems
Lecture24 – Ch 10. NoSQL

남 범 석

bnam@skku.edu

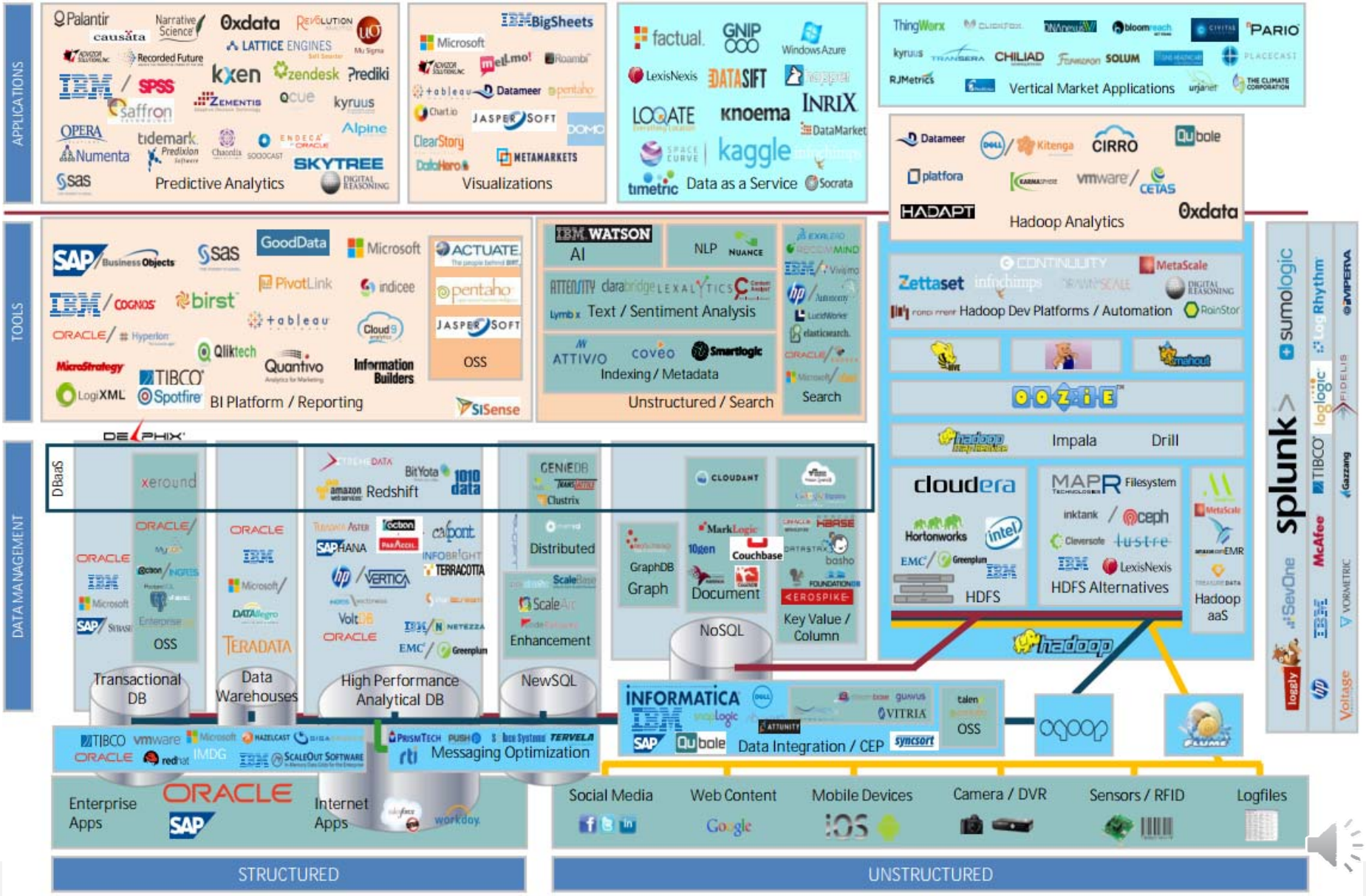
How to build a scalable system?

- Computer Science is a Science of Abstraction
  - creating the right model for a problem and devising the appropriate mechanizable techniques to solve it. – Alfred Aho



VS

Which solution to choose?



Big Data

- Transaction processing systems that need very high scalability
  - Many applications willing to sacrifice ACID properties and other database features, if they can get very high scalability
- Query processing systems that
  - Need very high scalability, and
  - Need to support non-relation data

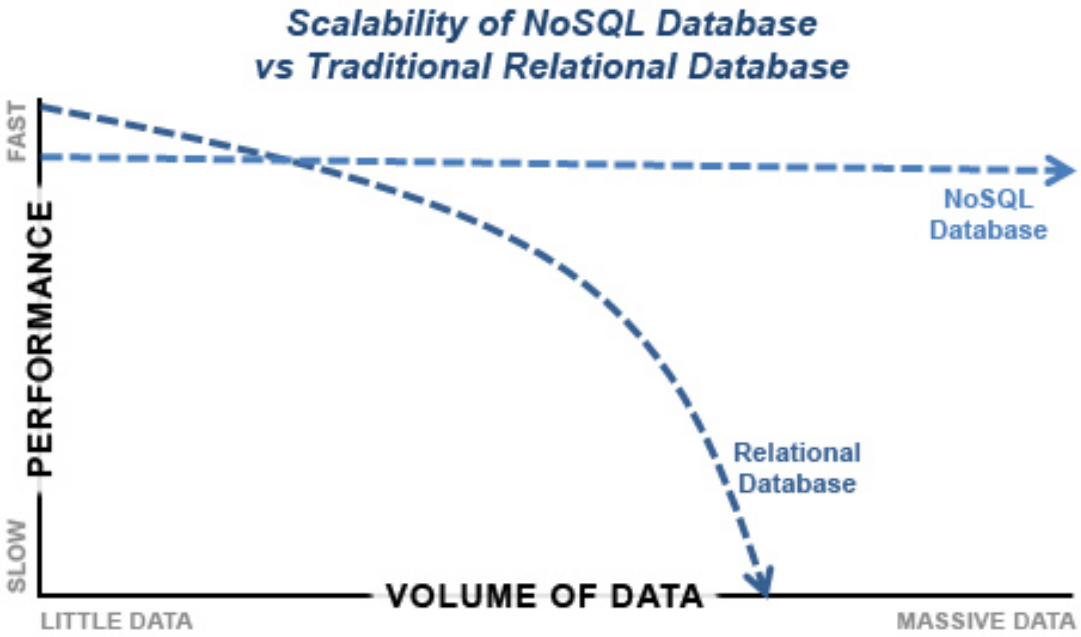
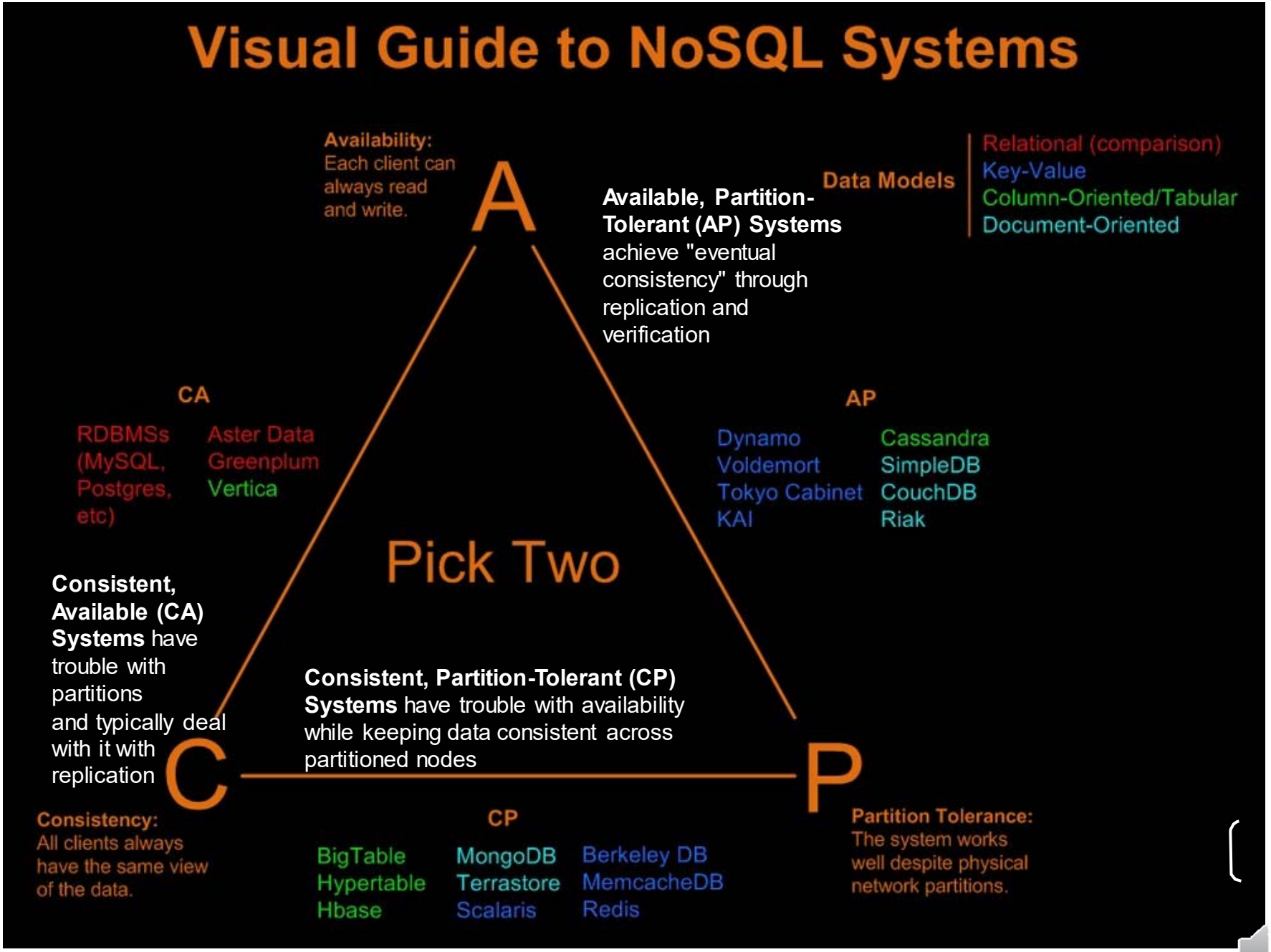


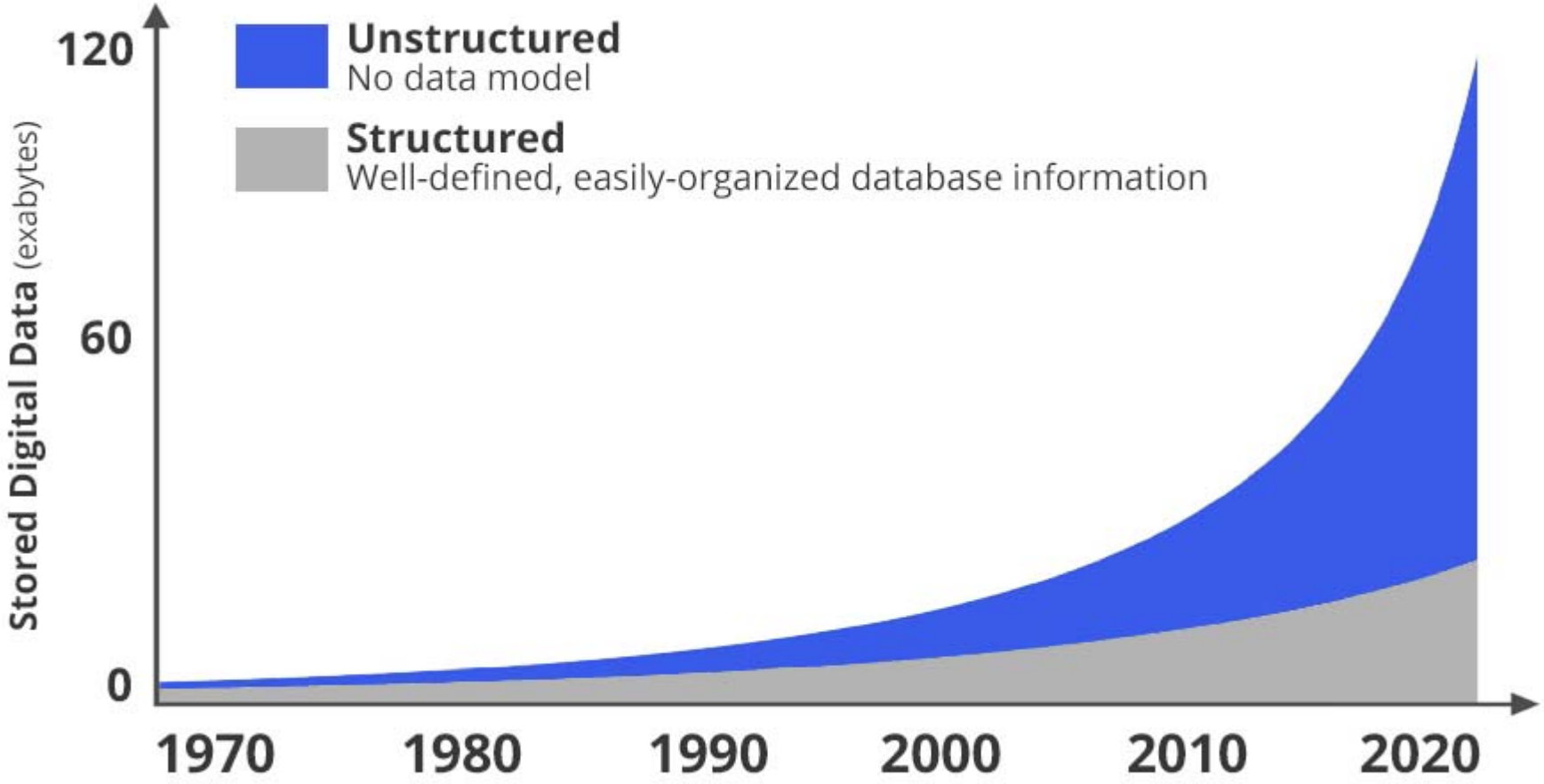
Image Credit: DataJobs.com



http://blog.nahurst.com/visual-guide-to-nosql-systems

Explosion of Unstructured/Semi-Structured Data

- A poor fit for the legacy RDBMS



Graph Source: IDC

Semi-Structured Data

- JSON: Textual representation widely used for data exchange

- Example 1:

```
{
        "ID": ”1111",
        "name": {
                "firstname: "Albert",
                "lastname: "Einstein"
        },
        "deptname": "Physics",
        "children": [
                {"firstname": "Hans", "lastname": "Einstein" },
                {"firstname": "Eduard", "lastname": "Einstein" }
        ]
}
```

- Example 2:

```
{
        "ID": "22222",
        "name": {
                ”Beomseok Nam"
        },
        "deptname": ”Computer Science",
        “e-mail”:“bnam@skku.edu”
}
```

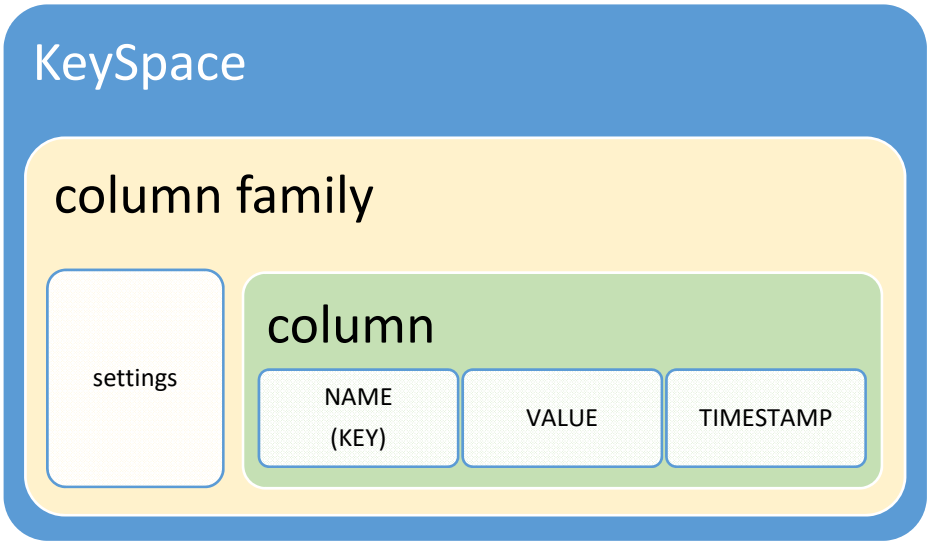
Data Model for Semi-Structured Data

- Structured Table for Semi-Structured Data?
  - Does not allow schema changes
  - Too many columns
  - Sparse tables

Structured Table (Schema)

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	7
10101	Srinivasan	Comp. Sci.	6
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Key-Value Stores (Schema-less)



Key-Value Storage (KVStore)

- KV-Stores seem very simple indeed
  - They are nothing but indexing structures
  - A simpler and more scalable "database"

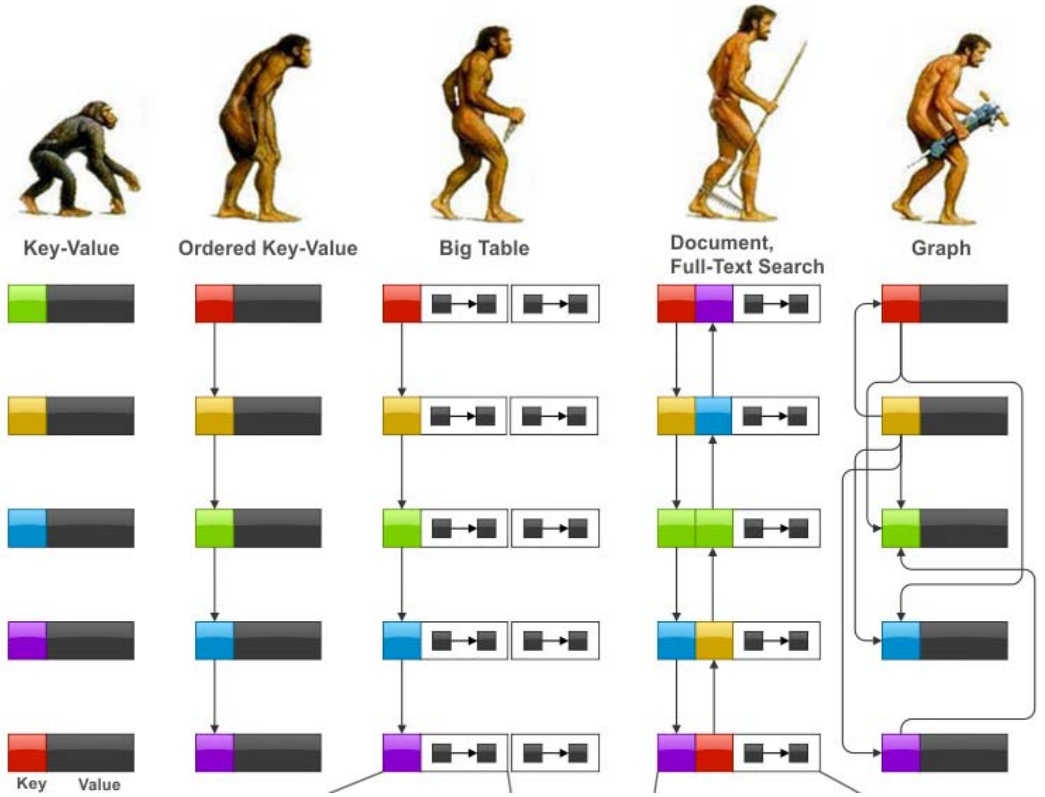
- Interface
  - `put(key, value);` // insert/write "value" associated with "key"
  - `value = get(key);` // get/read data associated with "key"

- Examples
  - Google BigTable & internal codes:
    - Key: hangoutID
    - Value: Hangout conversations

  - Facebook, Twitter:
    - Key: UserID
    - Value: user profile (e.g., posting history, photos, friends, …)

When to use Key-Value Stores over RDBMS?

- If data model is not complex nor hierarchical
- If workload is write-intensive
- If strong consistency is not required



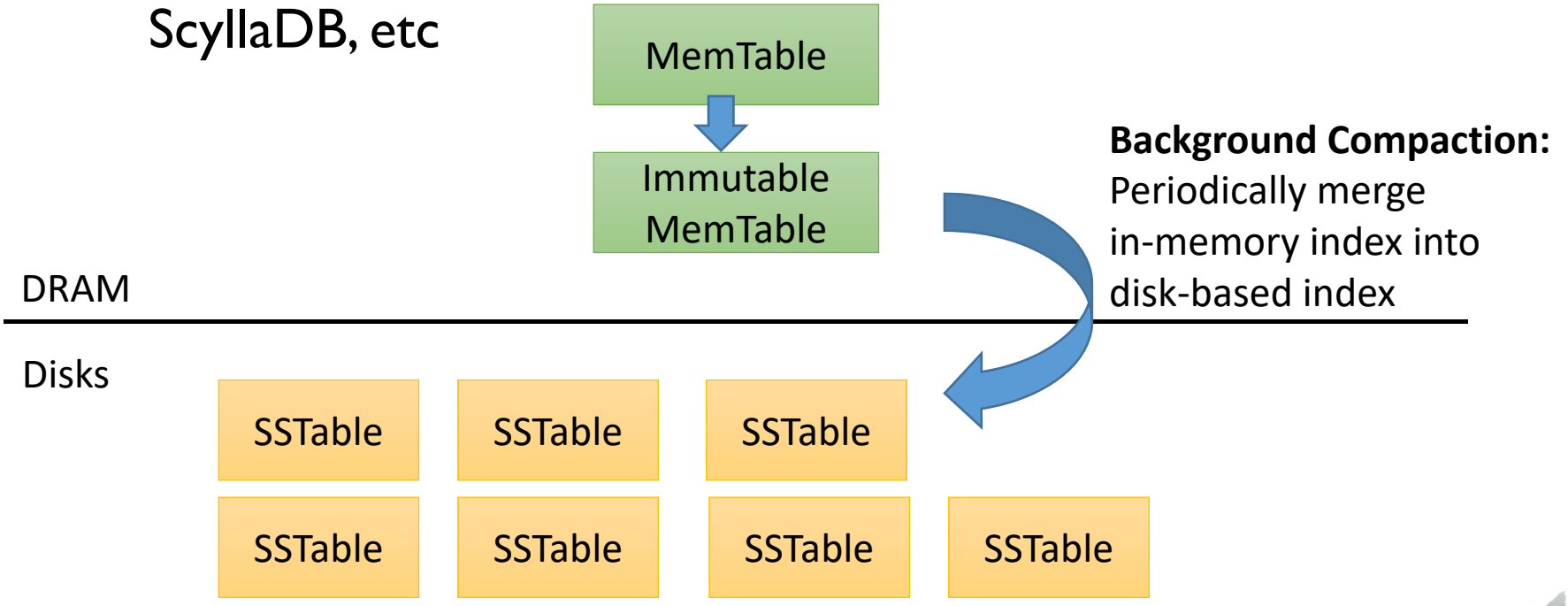
Full-featured
powerful
data management

Image Source: From RDBMS to Key-Value Store: Data Modeling Techniques- Wishmitha S. Mendis

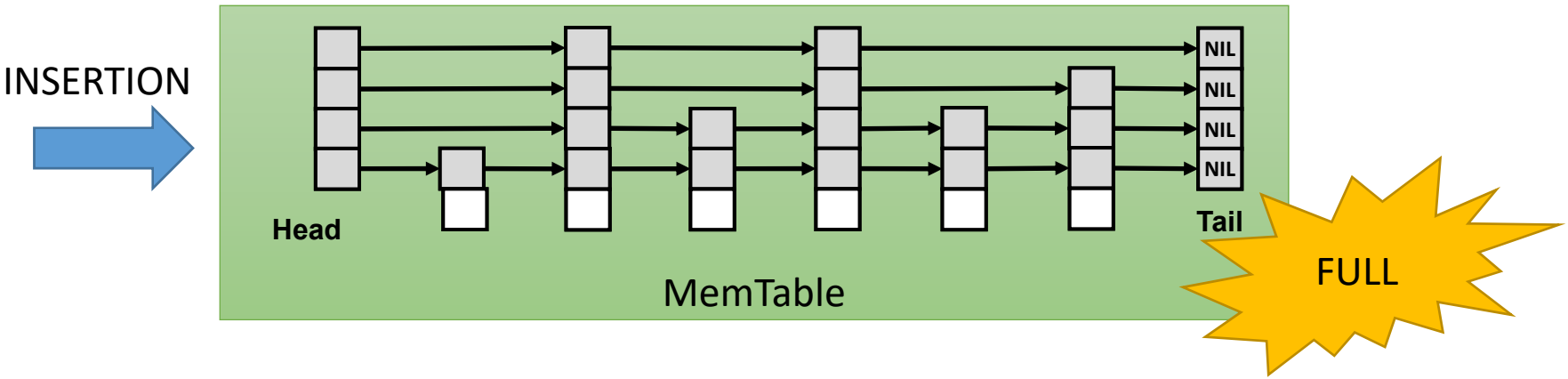
Chap. 24 Log-Structured Merge Tree

Background: Log-Structured Merge-Tree

- Designed for Write-intensive workloads
  - Fast insertion
  - Moderate search performance
- Widely used in various key-value stores
  - BigTable, Hbase, Cassandra, RocksDB, WiredTiger, InfluxDB, ScyllaDB, etc



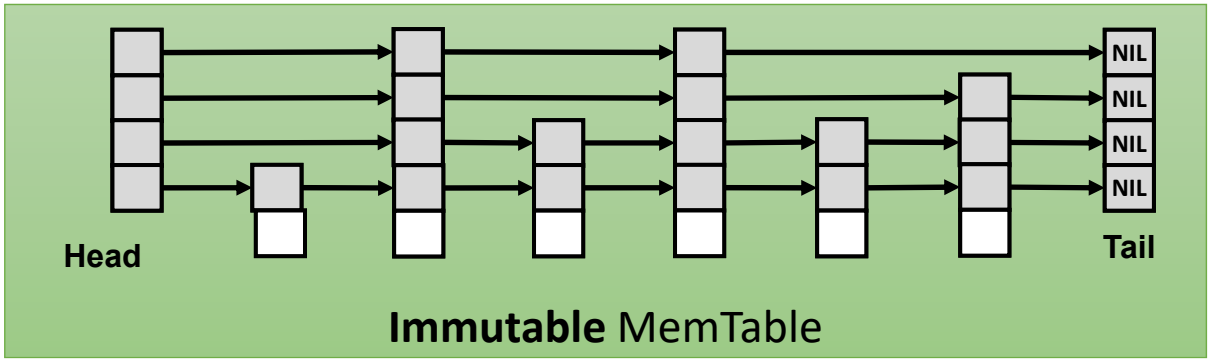
Background: Log-Structured Merge-Tree



DRAM

Disks

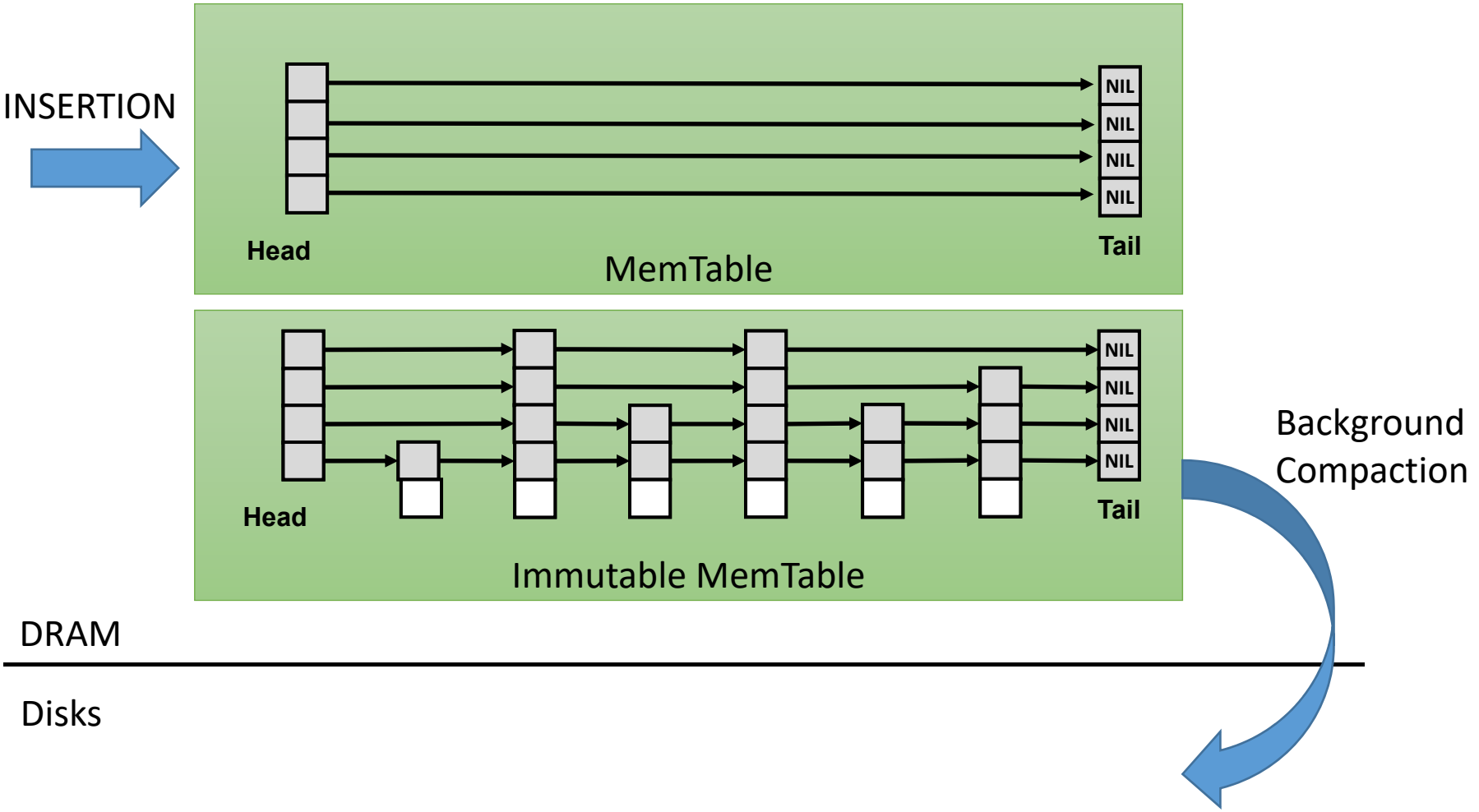
Background: Log-Structured Merge-Tree



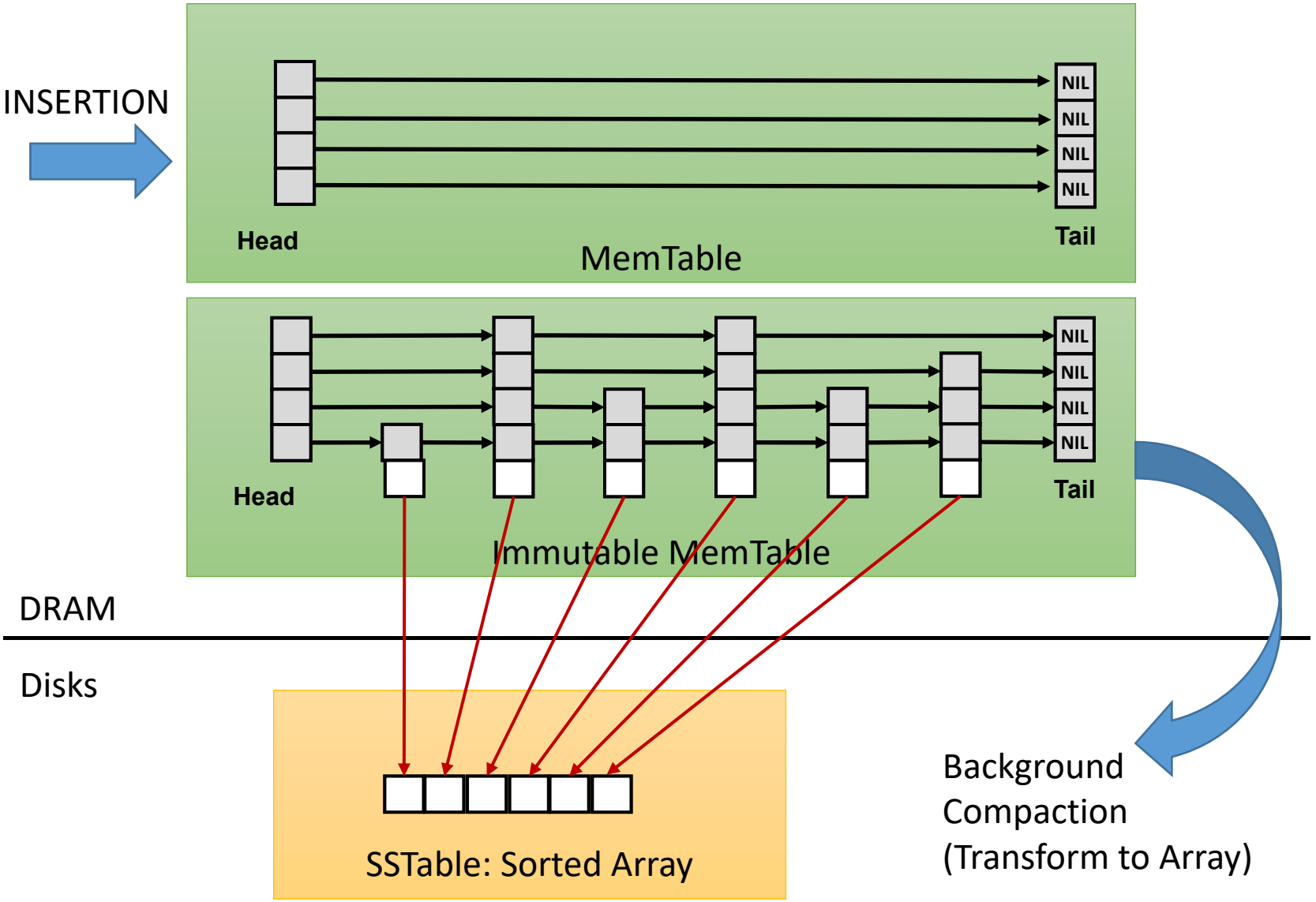
DRAM

Disks

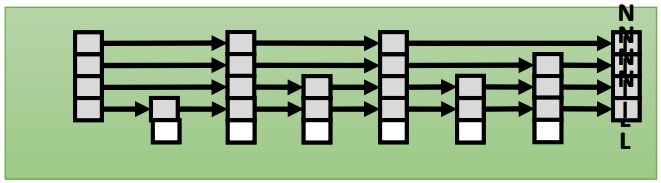
Background: Log-Structured Merge-Tree



Background: Log-Structured Merge-Tree



Background: Log-Structured Merge-Tree



MemTable

DRAM

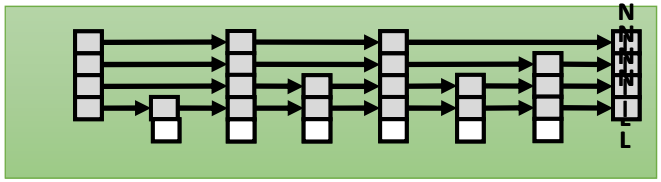
Disks

10, 15, 20, 25
SSTable

45, 55, 65, 75
SSTable

60, 80, 90, 100
SSTable

Background: Log-Structured Merge-Tree



MemTable

DRAM

Disks

Level 0

10, 15, 20, 25
SSTable

45, 55, 65, 75
SSTable

60, 80, 90, 100
SSTable

Level 1

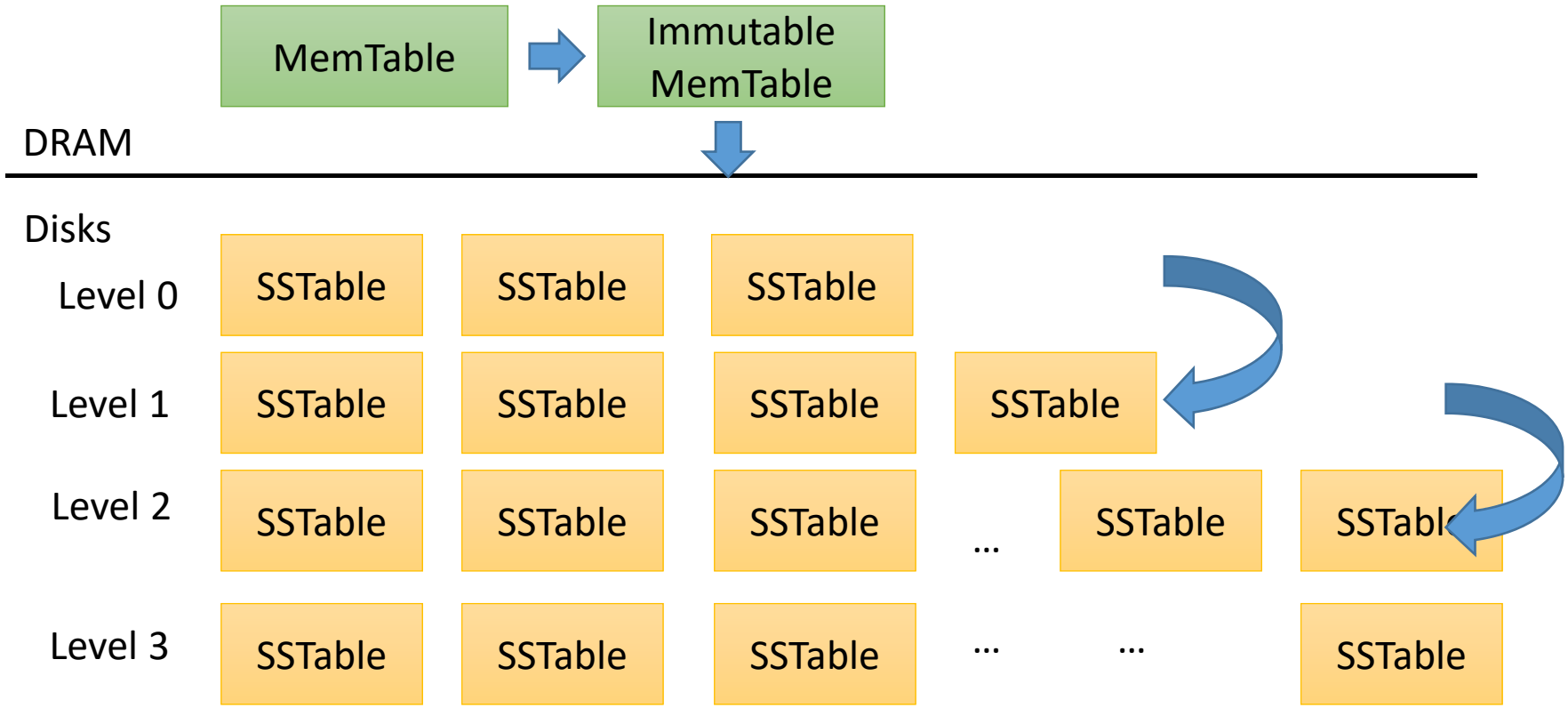
45, 55, 60, 65
SSTable

75, 80, 90, 100
SSTable

Background
Compaction
(Merge Sort)

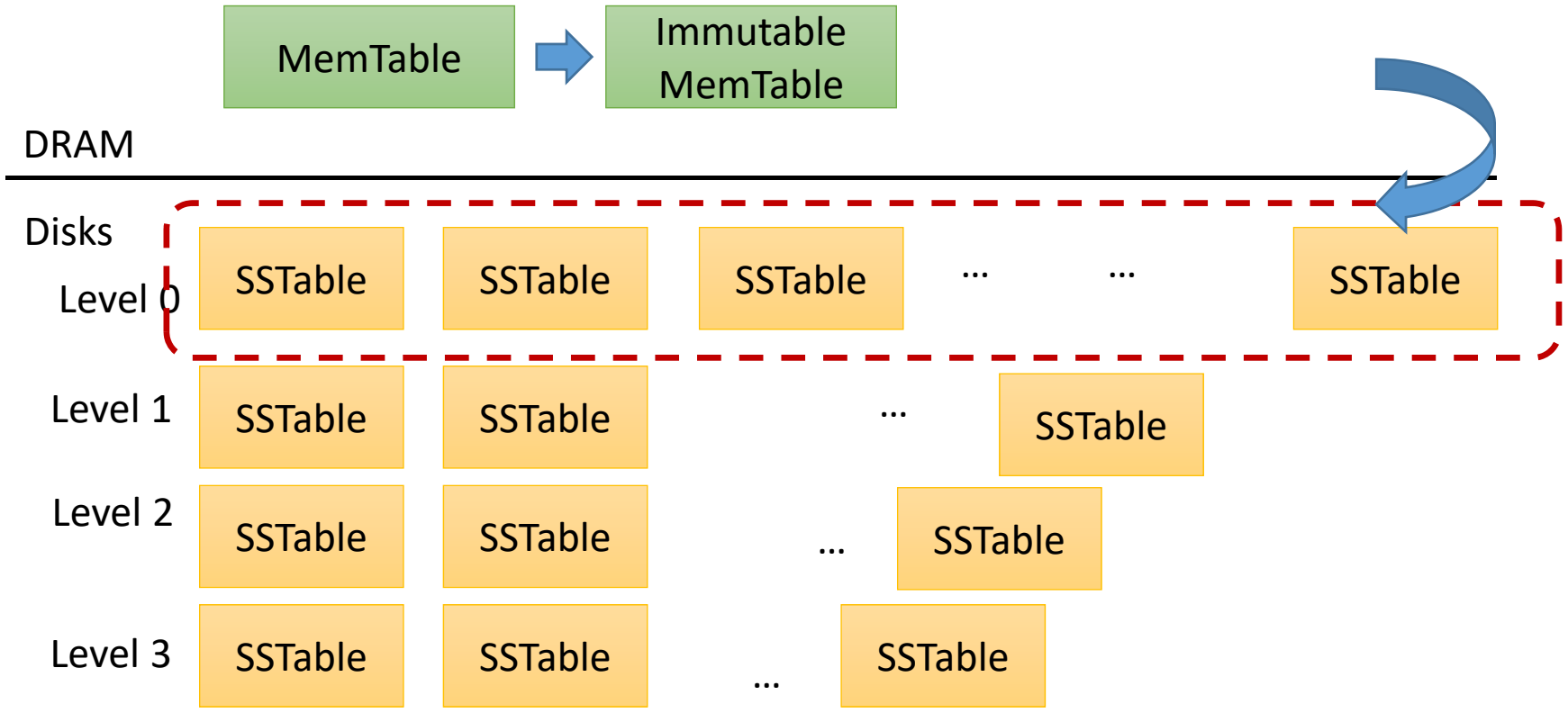
Same key-values are written multiple times
→ Write Amplification Problem

Background: Log-Structured Merge-Tree



This is what LSM-Tree should be

Background: Log-Structured Merge-Tree



But, In Reality

[1] Memory writes are very fast
[2] Merge sort on disks is slow

Background: Log-Structured Merge-Tree

No More Writes to MemTable !! → Write Stall Problem

But, In Reality

[1] Memory writes are very fast
[2] Merge sort on disks is slow

MongoDB

RDB ACID to NoSQL BASE

Atomicity

Consistency

Isolation

Durability

⟺

Basically

Available (CP)

Soft-state
(State of system may change over time)

Eventually consistent
(Asynchronous propagation)

Key Value Storage Systems

- Key-value storage systems store large numbers (billions or even more) of small (KB-MB) sized records
- Records are **partitioned** across multiple machines and
- Queries are routed by the system to appropriate machine
- Records are also **replicated** across multiple machines, to ensure availability even if a machine fails
  - Key-value stores ensure that updates are applied to all replicas, to ensure that their values are **consistent**

What is MongoDB?

- Developed by 10gen
  - Founded in 2007
- A document-oriented, NoSQL database
  - Hash-based, s*chema-less database*
    - No Data Definition Language
    - In practice, this means you can store hashes with any keys and values that you choose
      - Keys are stored as strings
      - Document Identifiers (_id) will be created for each document, field name reserved by system
    - Application tracks the schema and mapping
    - Uses BSON format
      - Based on JSON – B stands for Binary

Functionality of MongoDB

- Dynamic schema
  - No DDL
- Document-based database
- Secondary indexes
- Query language via an API
- Atomic writes and fully-consistent reads
  - If system configured that way
- Master-slave replication with automated failover (replica sets)
- Built-in horizontal scaling via automated range-based partitioning of data (sharding)
- No joins nor transactions

Why use MongoDB?

- Simple queries
- Functionality provided applicable to most web applications
- Easy and fast integration of data
  - No ERD diagram
- Not well suited for heavy and complex transactions systems

Data Model

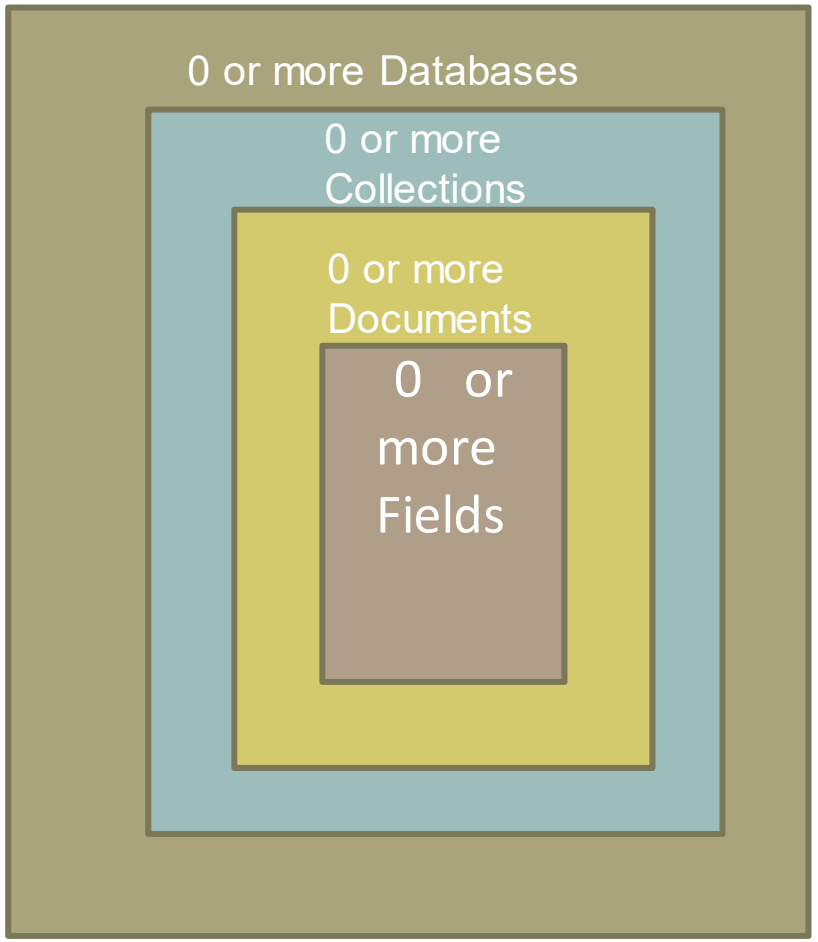
- Stores data in form of BSON (Binary JSON) *documents*

```
{
        name: “travis”,
        salary: 30000,
        designation: “Computer Scientist”,
        teams: [ “front-end”,      “database” ]
}
```

- Group of related *documents* with a shared common index is a *collection*

MongoDB: Hierarchical Objects

- A MongoDB instance may have zero or more ‘databases’
- A database may have zero or more ‘collections’.
- A collection may have zero or more ‘documents’.
- A document may have one or more ‘fields’.
- MongoDB ‘Indexes’ function much like their RDBMS counterparts.



RDB Concepts to NO SQL

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

Collection is not strict about what it Stores

Schema-less

Hierarchy is evident in the design

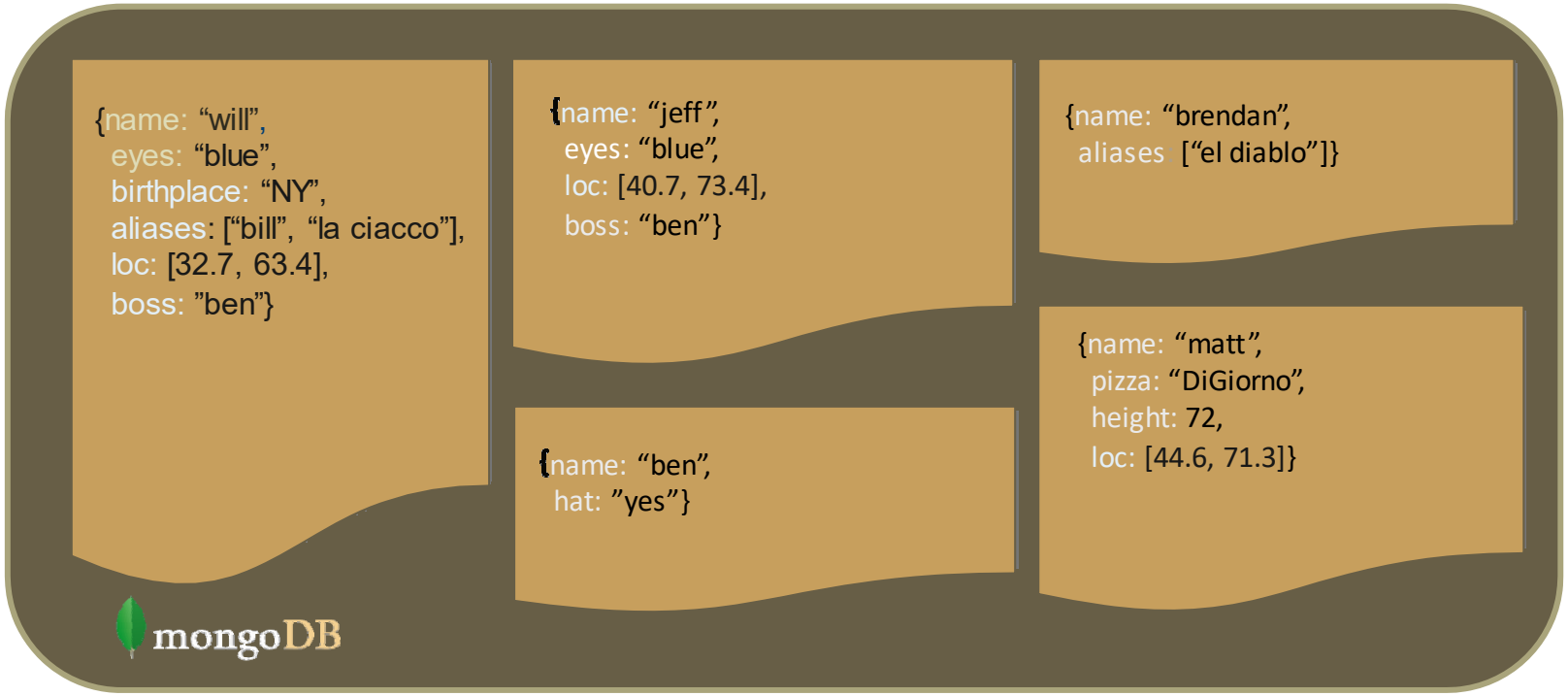
Embedded Document ?

MongoDB Processes and configuration

- Mongod – Database instance
- Mongos - Sharding processes
  - Analogous to a database router.
  - Processes all requests
  - Decides how many and which *mongod*s should receive the query
  - *Mongos* collates the results, and sends it back to the client.
- Mongo – an interactive shell (a client)
  - Fully functional JavaScript environment for use with a MongoDB
- You can have one *mongos* for the whole system no matter how many mongods you have
- OR you can have one local *mongos* for every client if you wanted to minimize network latency.

Schema Free

- MongoDB does not need any pre-defined data schema
- Every document in a collection could have different data
  - Addresses NULL data fields

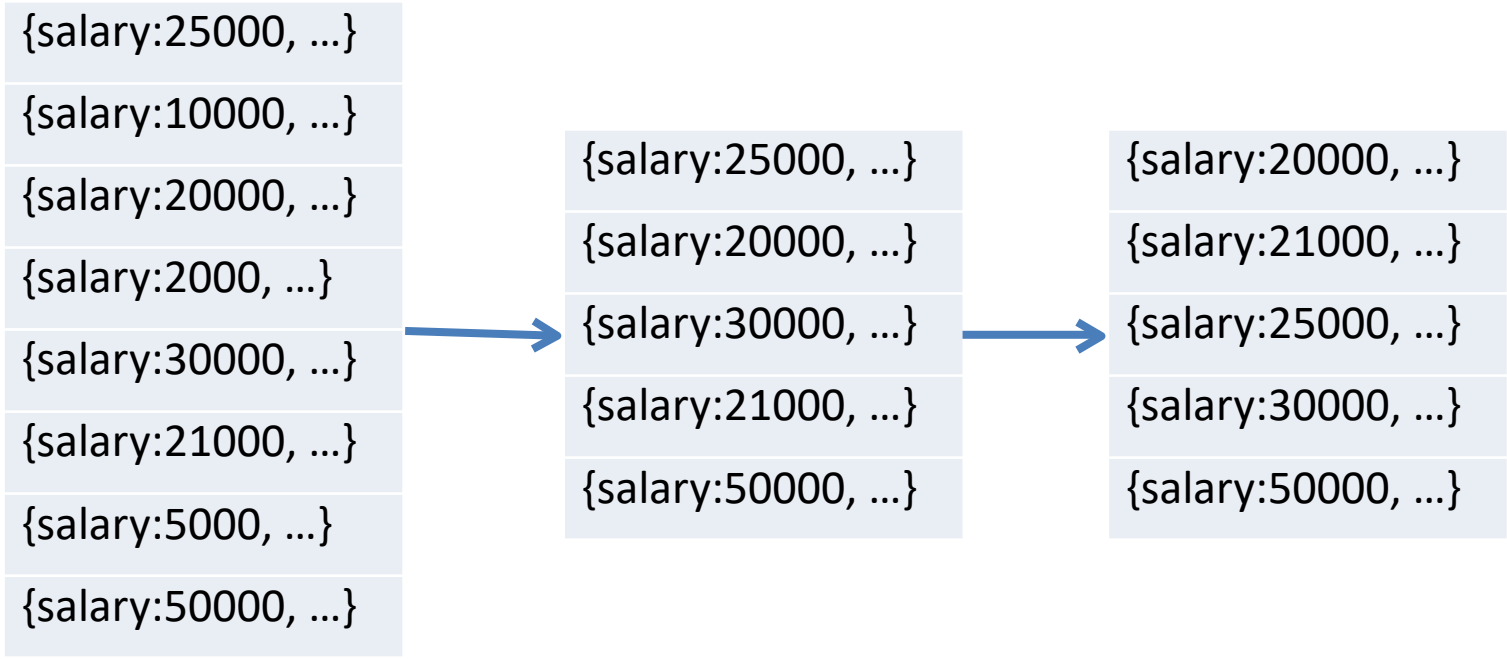


Query

- Query all employee names with salary greater than 18000 sorted in ascending order

```
db.users.find({salary:{$gt:18000}, {name:1}}).sort({salary:1})
```

Collection — Condition — Projection — Modifier



Query Operators

Name	Description
$eq	Matches value that are equal to a specified value
$gt, $gte	Matches values that are greater than (or equal to a specified value
$lt, $lte	Matches values less than or (equal to) a specified value
$ne	Matches values that are not equal to a specified value
$in	Matches any of the values specified in an array
$nin	Matches none of the values specified in an array
$or	Joins query clauses with a logical OR returns all
$and	Join query clauses with a loginal AND
$not	Inverts the effect of a query expression
$nor	Join query clauses with a logical NOR
$exists	Matches documents that have a specified field

https://docs.mongodb.org/manual/reference/operator/query/

Insert

- Insert a row entry for new employee Sally

```
db.users.insert({
        name: “sally”,
        salary: 15000,
        designation: “MTS”,
        teams: [ “cluster-management” ]
        })
```

Update

- All employees with salary greater than 18000 get a designation of Executive

```
db.users.update(
    {salary:{$gt:18000}},
    {$set: {designation: “Manager”}},
    {multi: true}
)
```

Update Criteria

Update Action

Update Option

- Multi option allows multiple document update

Delete

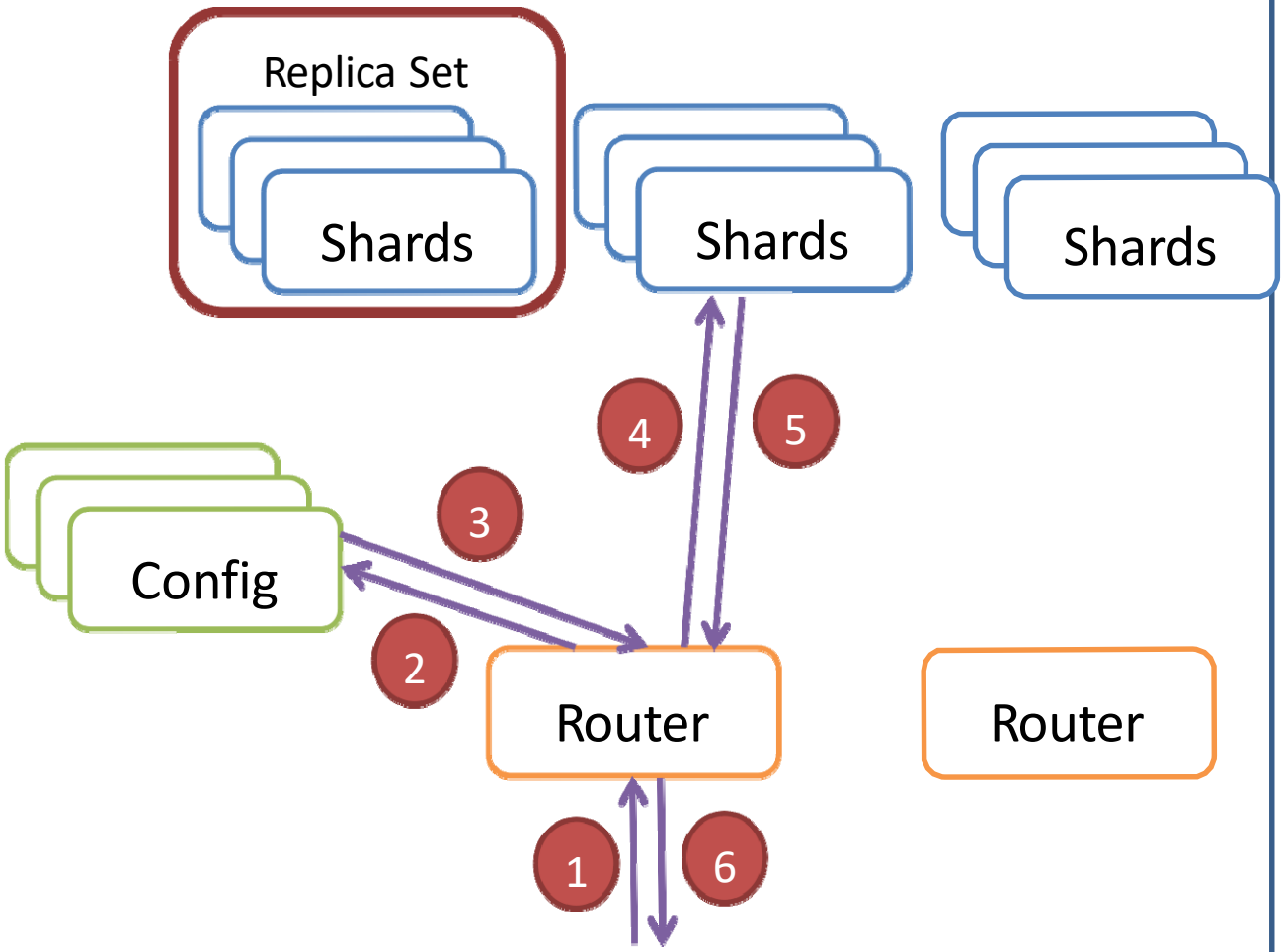
- Remove all employees who earn less than 10000

Remove Criteria

```
db.users.remove(
    {salary:{$lt:10000}},
)
```

- Can accept a flag to limit the number of document removal

Typical MongoDB Deployment



- Shards: mongod servers store the data
- Multiple shard servers form a *replica set*
- Replica set maintain same replica of data
- Routers: mongos interfaces with clients and routers operations to appropriate shards
- Config: Stores collection level metadata.

Read Preference

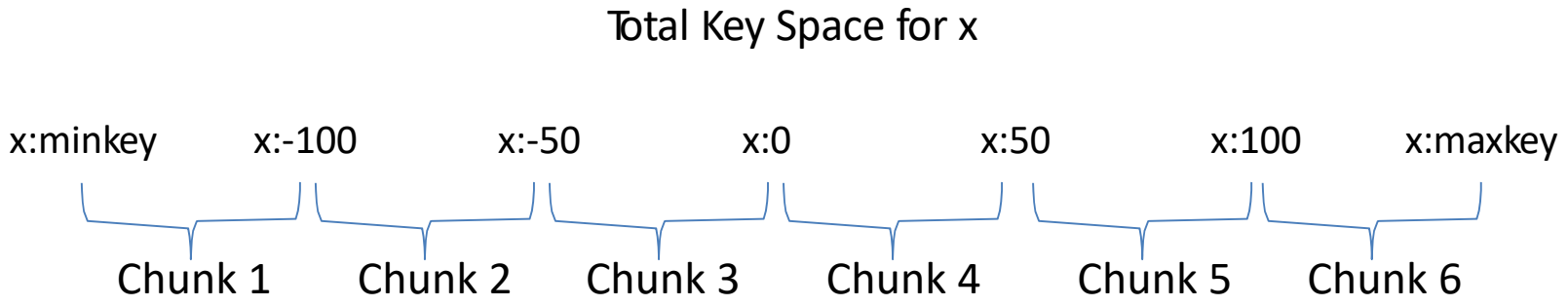
- Determine where to route read operation
- Default is primary. Possible options are secondary, primary-preferred, etc.
- Helps reduce latency, improve throughput
- Reads from secondary may fetch stale data

Write Concern

- Determines the guarantee that MongoDB provides on the success of a write operation
- Default is *acknowledged.* Others are unacknowledged, replica-acknowledged, etc
- For the default case, primary replicas acknowledge the success of a write operation
- Weaker write concern implies faster write time

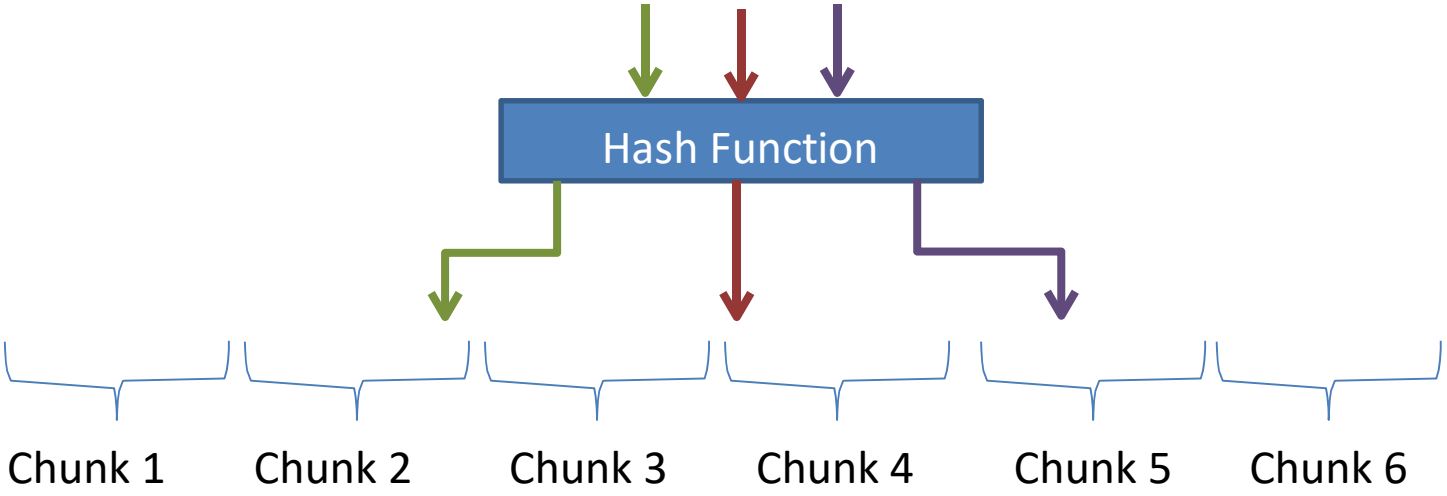
Partition

- Shard Key: Single or compound field in schema used for data partitioning
- Partitions are called *chunks.* Two strategies:
  - [1] Range based: Shard Key Values are partitioned into ranges



Partition

- [2] Hash based: Hash of shard key values are partitioned into ranges

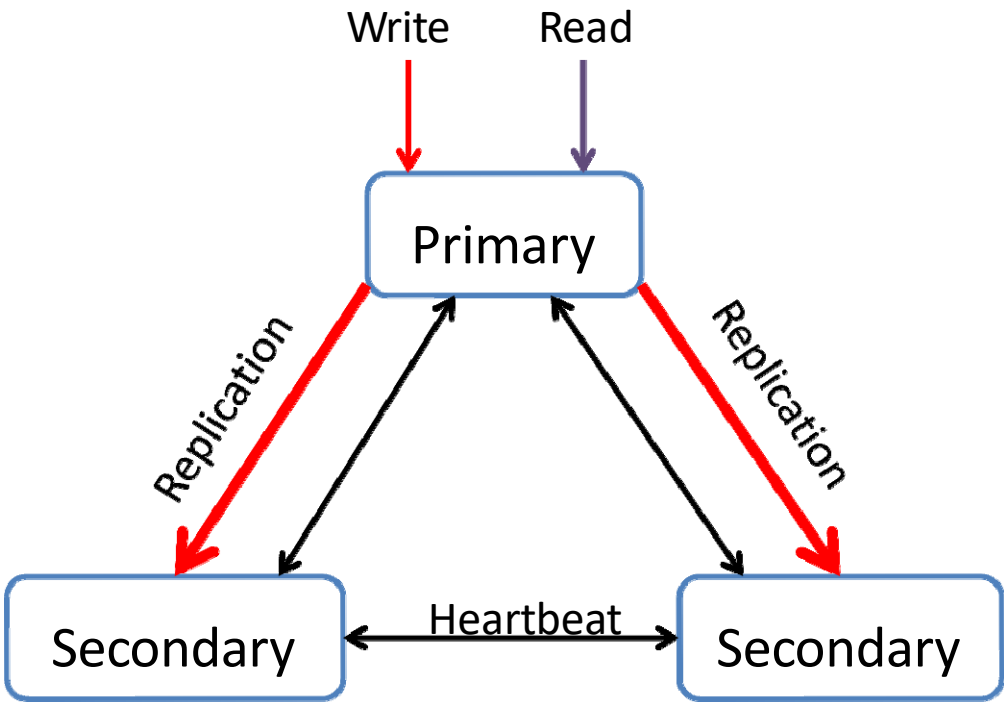


- Range Queries are efficient for the first strategy
- Hash Scheme leads to better data balancing

Balancing

- Splitting: Background process which splits when a chunks grows beyond a threshold
- Balancing: Migrates chunks among shards if there is an uneven distribution

Replication



Consistency

- Strongly Consistent: Read Preference is Master
- Eventually Consistent: Read Preference is Slave
- CAP Theorem: Under partition, MongoDB becomes write unavailable thereby ensuring consistency