Data Management for Data Science

Lecture 12: NoSql and KeyValue stores

Prof. Asoc. Endri Raço

Today’s Lecture

1. Intro to NoSQL

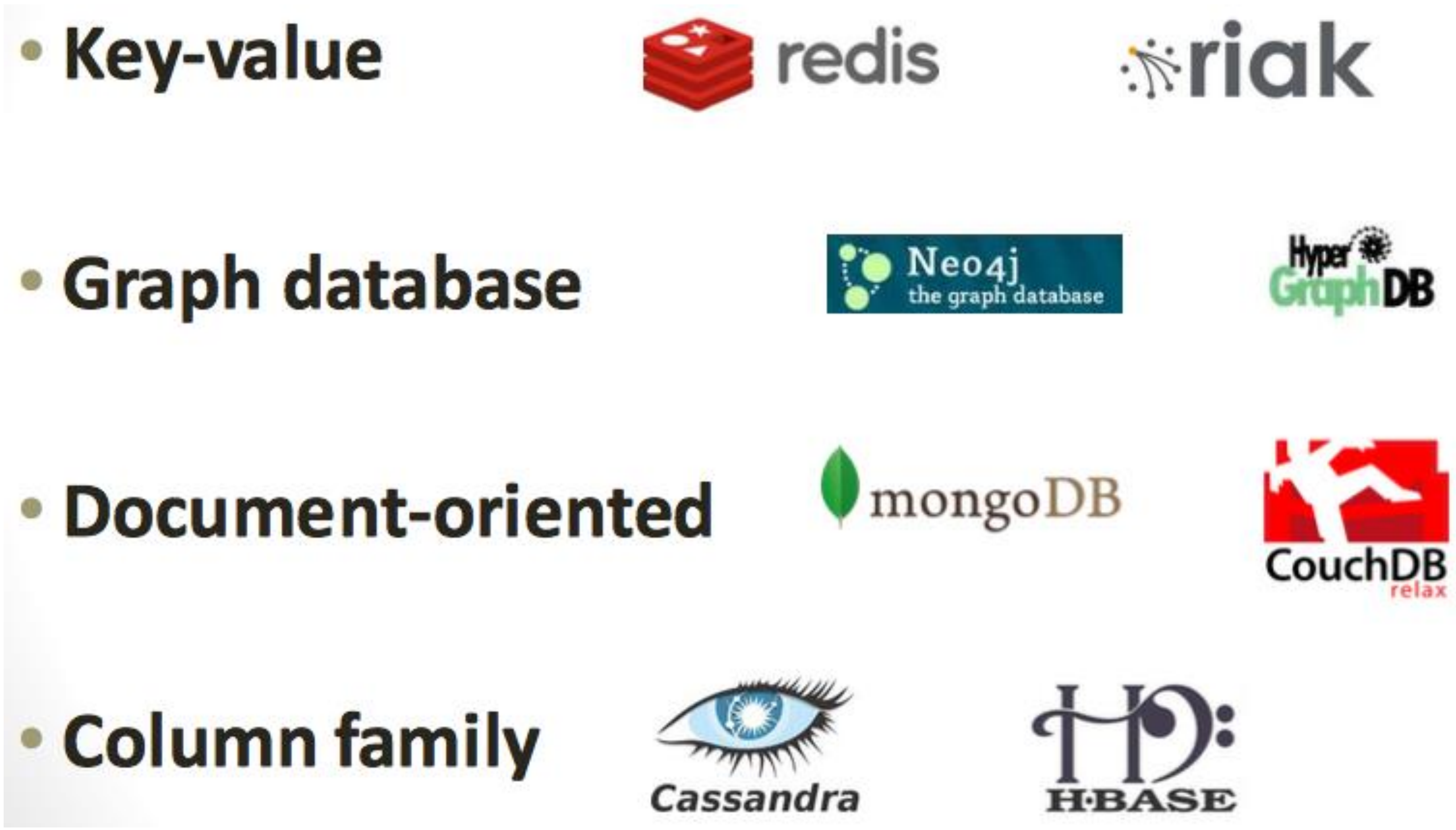
2. NoSQL Assumptions and the CAP Theorem

3. Strengths and weaknesses of NoSQL

4. Example: MongoDB

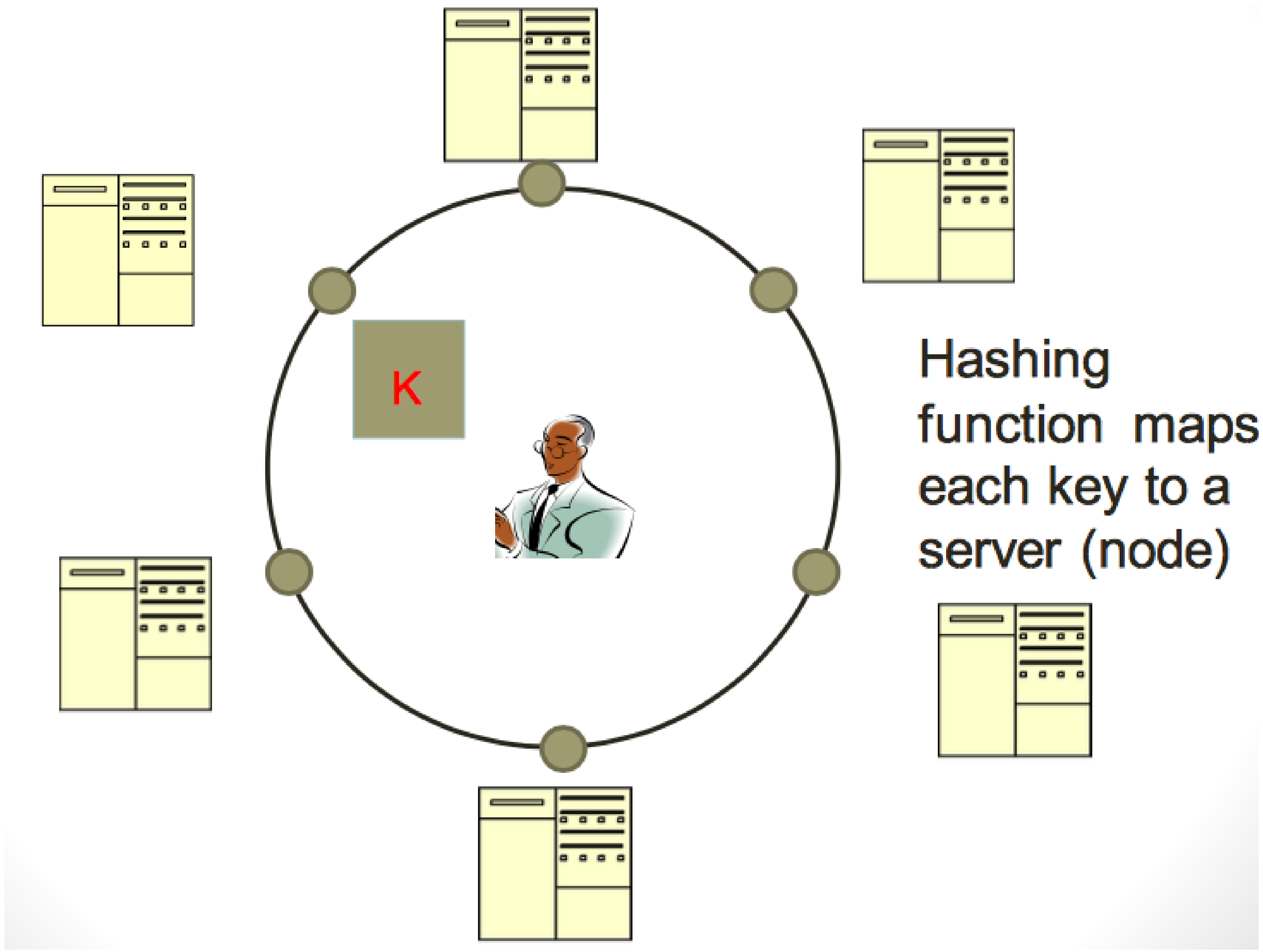
1. Intro to NoSQL

Taxonomy of NoSQL



- Key-value — redis, riak
- Graph database — Neo4j the graph database, HyperGraphDB
- Document-oriented — mongoDB, CouchDB relax
- Column family — Cassandra, H·BASE

Typical NoSQL architecture



Hashing function maps each key to a server (node)

2. NoSQL Assumptions and the CAP Theorem

CAP theorem for NoSQL

- What the CAP theorem really says: If you cannot limit the number of faults and requests can be directed to any server and you insist on serving every request you receive then you cannot possibly be consistent

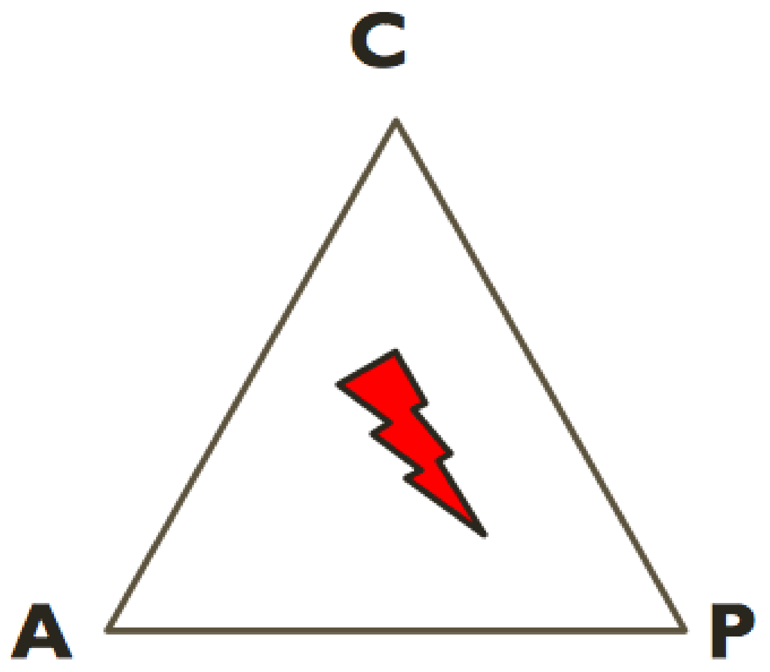
- How it is interpreted: You must always give something up: consistency, availability or tolerance to failure and reconfiguration

CAP theorem for NoSQL

GIVEN:

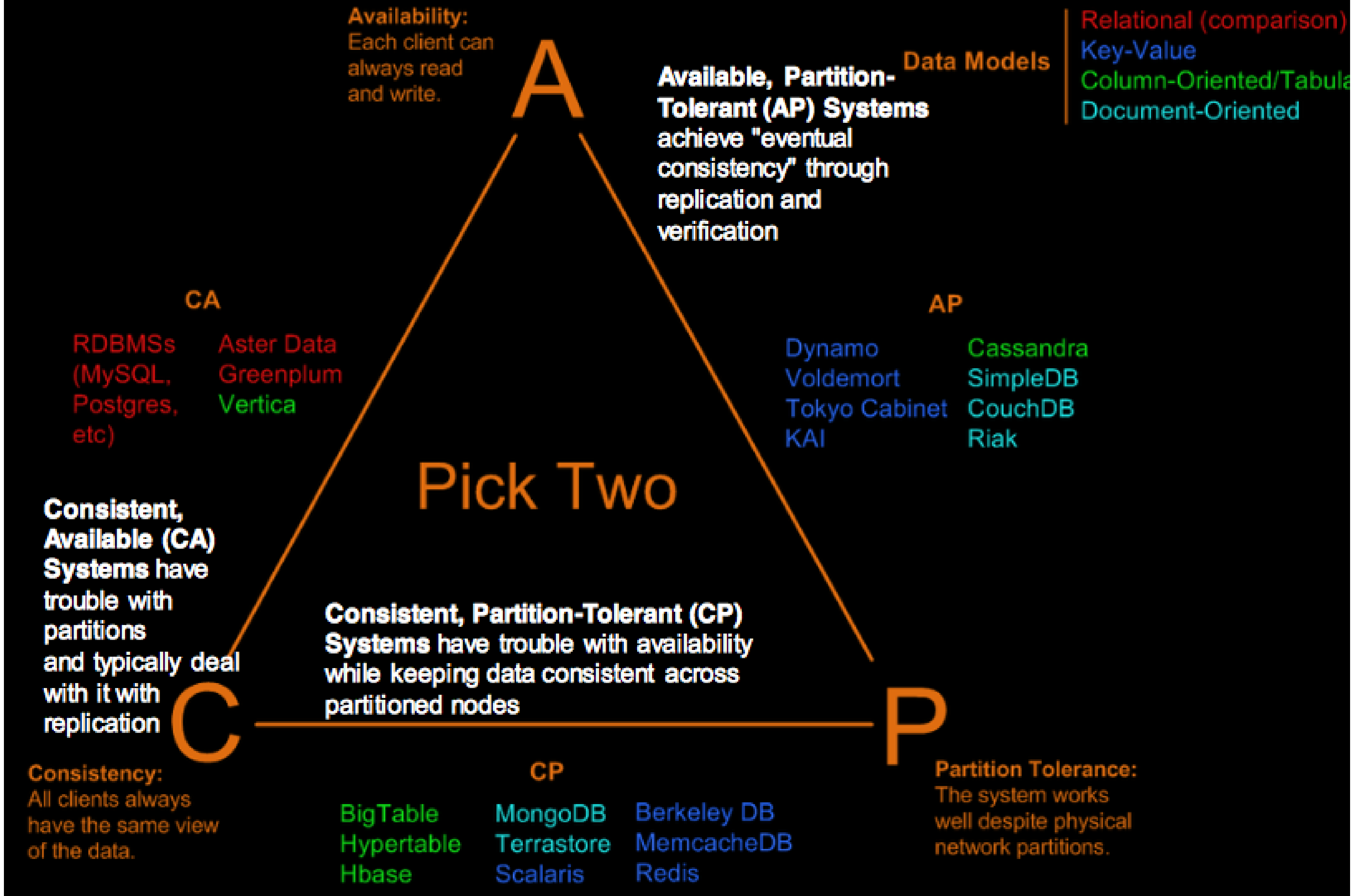
- Many nodes
- Nodes contain ***replicas of partitions*** of the data

- Consistency
  - All replicas contain the same version of data
  - Client always has the same view of the data (no matter what node)
- Availability
  - System remains operational on failing nodes
  - All clients can always read and write
- Partition tolerance
  - multiple entry points
  - System remains operational on system split (communication malfunction)
  - System works well across physical network partitions



CAP Theorem:
satisfying all three at the same time is impossible

Visual Guide to NoSQL Systems



Availability: Each client can always read and write.

Available, Partition-Tolerant (AP) Systems achieve "eventual consistency" through replication and verification

Data Models
- Relational (comparison)
- Key-Value
- Column-Oriented/Tabula
- Document-Oriented

CA

RDBMSs (MySQL, Postgres, etc)	Aster Data
	Greenplum
	Vertica

AP

Dynamo	Cassandra
Voldemort	SimpleDB
Tokyo Cabinet	CouchDB
KAI	Riak

Consistent, Available (CA) Systems have trouble with partitions and typically deal with it with replication

Consistent, Partition-Tolerant (CP) Systems have trouble with availability while keeping data consistent across partitioned nodes

Consistency: All clients always have the same view of the data.

CP

BigTable	MongoDB	Berkeley DB
Hypertable	Terrastore	MemcacheDB
Hbase	Scalaris	Redis

Partition Tolerance: The system works well despite physical network partitions.

Sharding of data

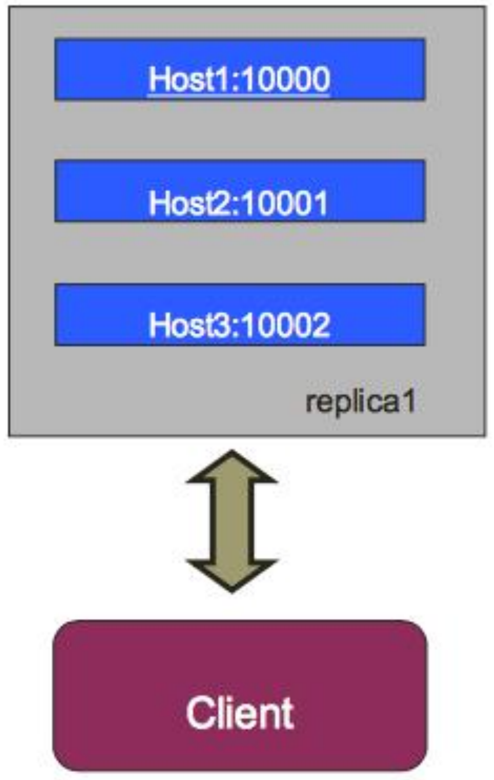
- Distributes a single logical database system across a cluster of machines
- Uses range-based partitioning to distribute documents based on a specific shard key
- Automatically balances the data associated with each shard
- Can be turned on and off per collection (table)

Replica Sets

- Redundancy and Failover
- Zero downtime for upgrades and maintenance

- Master-slave replication
  - Strong Consistency
  - Delayed Consistency

- Geospatial features

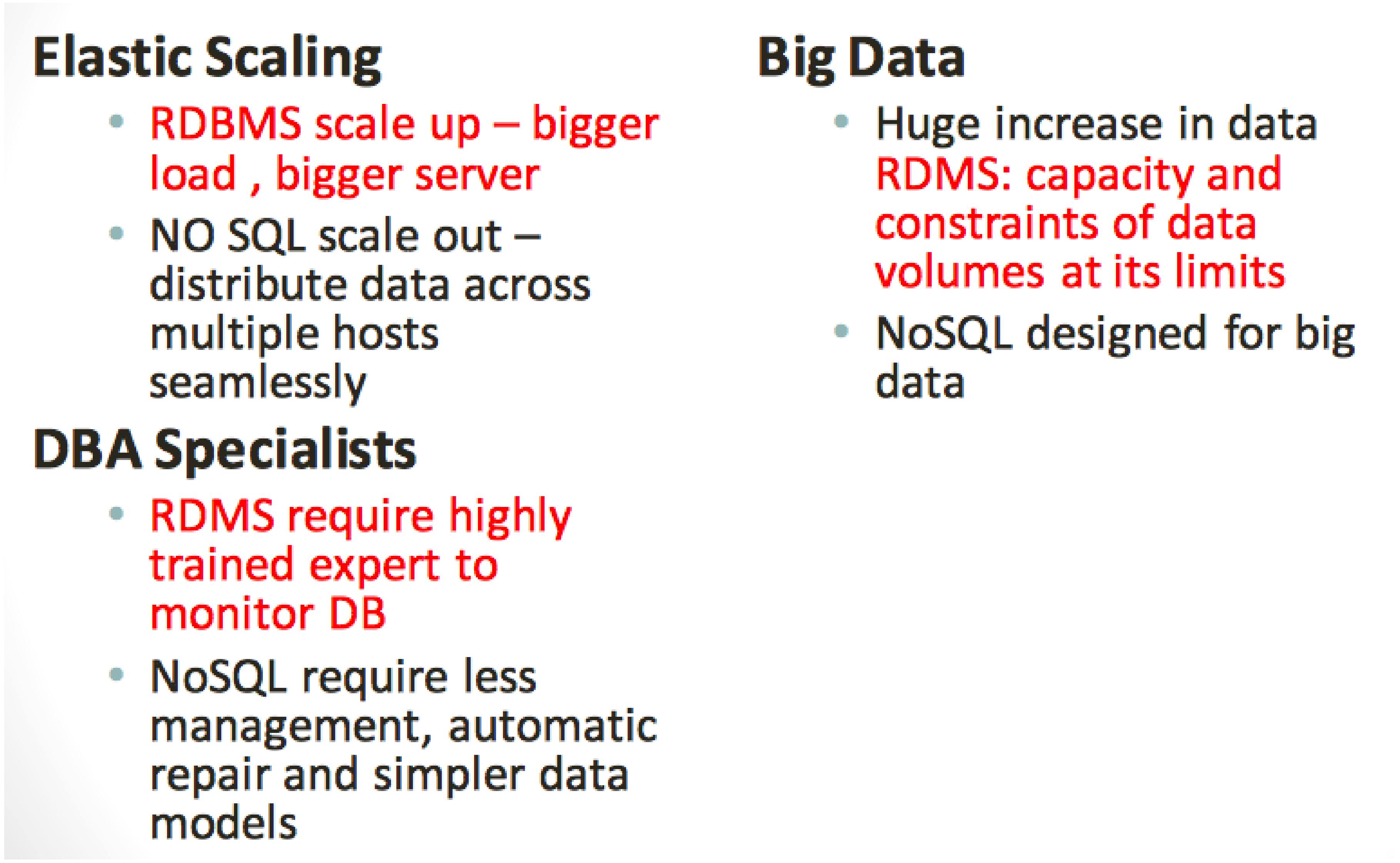


3. Strengths and weaknesses of NoSQL

How does NoSQL vary from RDBMS?

- Looser schema definition
- Applications written to deal with specific documents/ data
  - Applications aware of the schema definition as opposed to the data
- Designed to handle distributed, large databases
- Trade offs:
  - No strong support for ad hoc queries but designed for speed and growth of database
    - Query language through the API
  - Relaxation of the ACID properties

Benefits of NoSQL



Elastic Scaling

- RDBMS scale up – bigger load , bigger server
- NO SQL scale out – distribute data across multiple hosts seamlessly

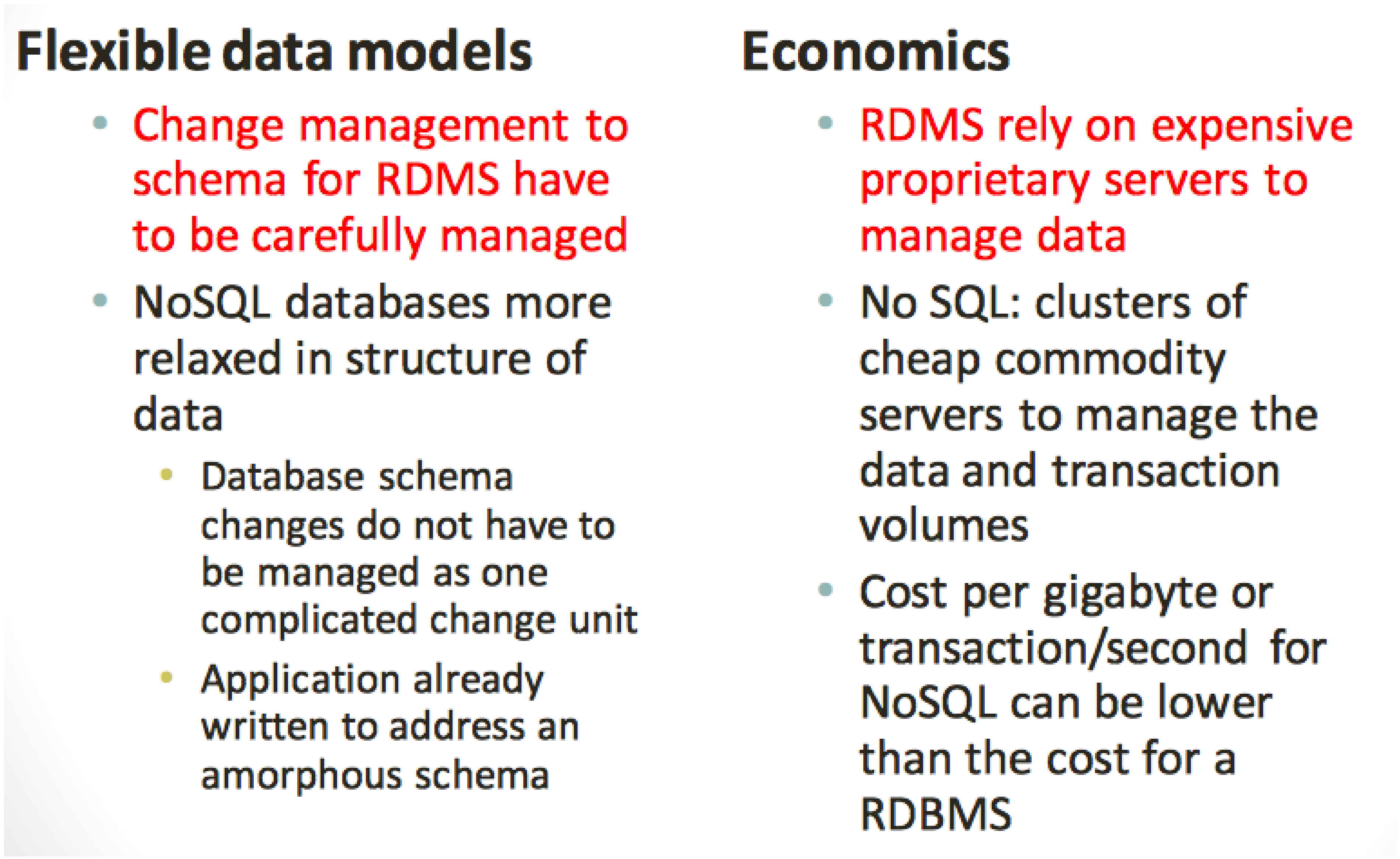
DBA Specialists

- RDMS require highly trained expert to monitor DB
- NoSQL require less management, automatic repair and simpler data models

Big Data

- Huge increase in data RDMS: capacity and constraints of data volumes at its limits
- NoSQL designed for big data

Benefits of NoSQL



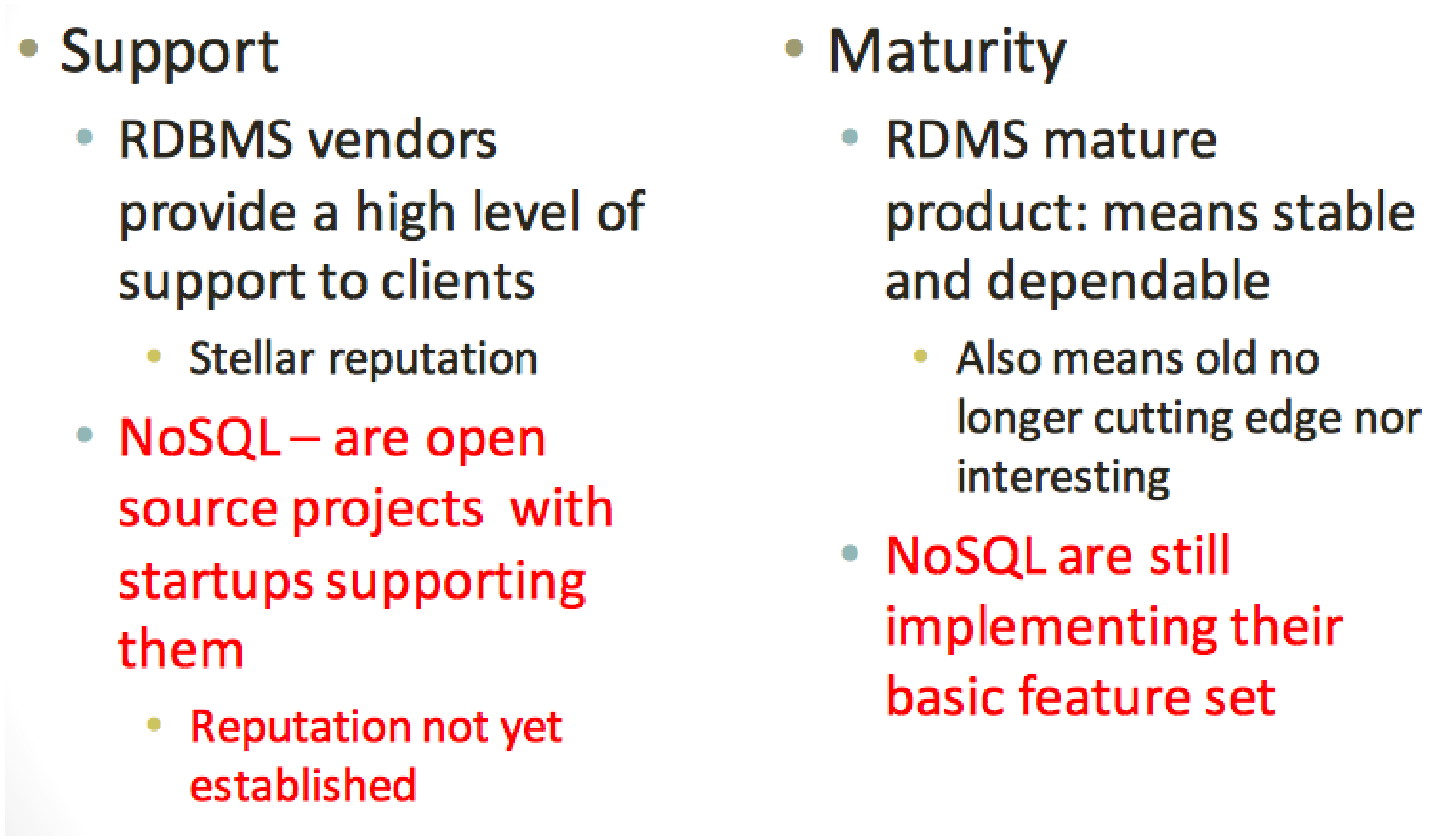
Flexible data models

- Change management to schema for RDMS have to be carefully managed
- NoSQL databases more relaxed in structure of data
  - Database schema changes do not have to be managed as one complicated change unit
  - Application already written to address an amorphous schema

Economics

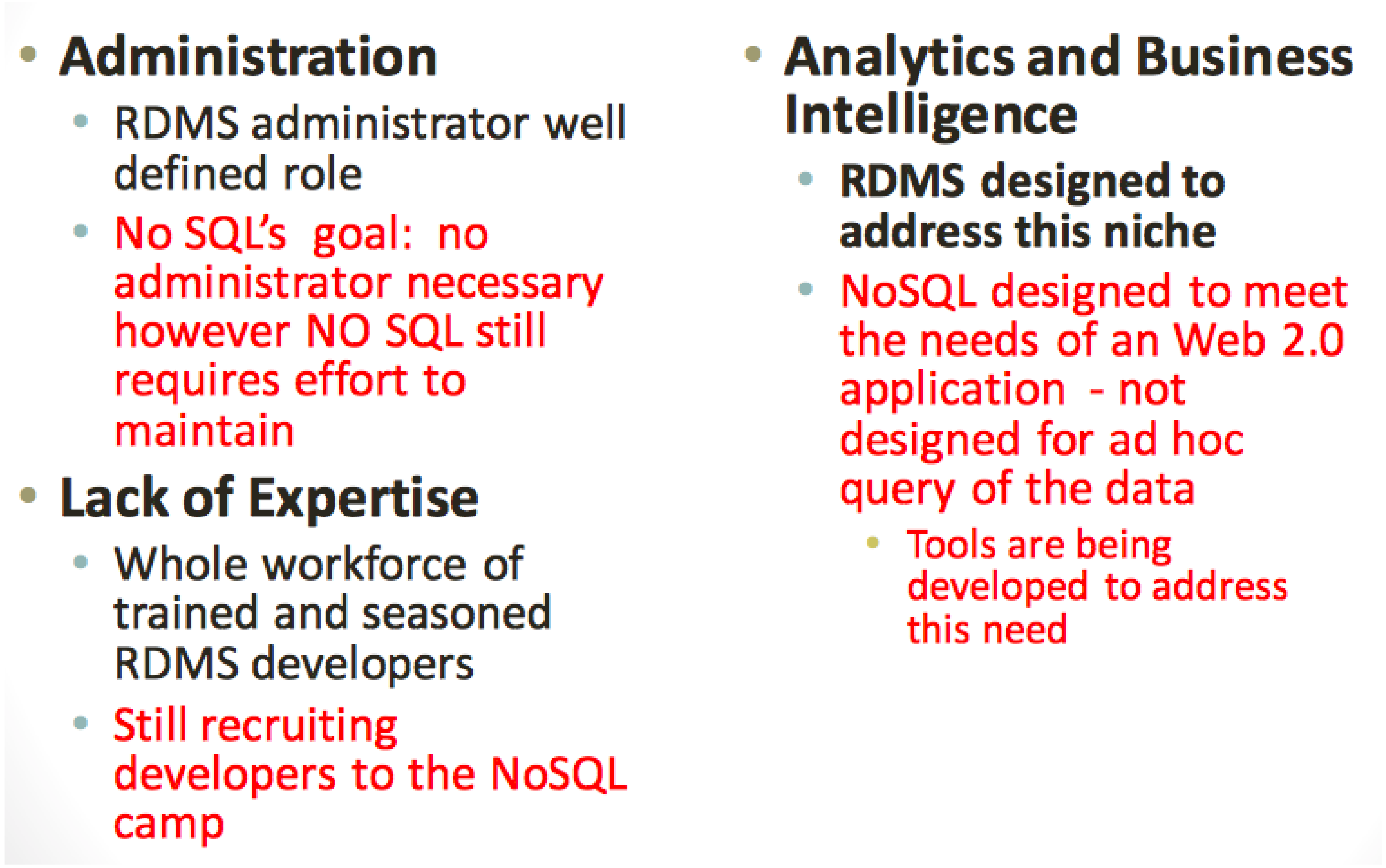
- RDMS rely on expensive proprietary servers to manage data
- No SQL: clusters of cheap commodity servers to manage the data and transaction volumes
- Cost per gigabyte or transaction/second for NoSQL can be lower than the cost for a RDBMS

Drawbacks of NoSQL



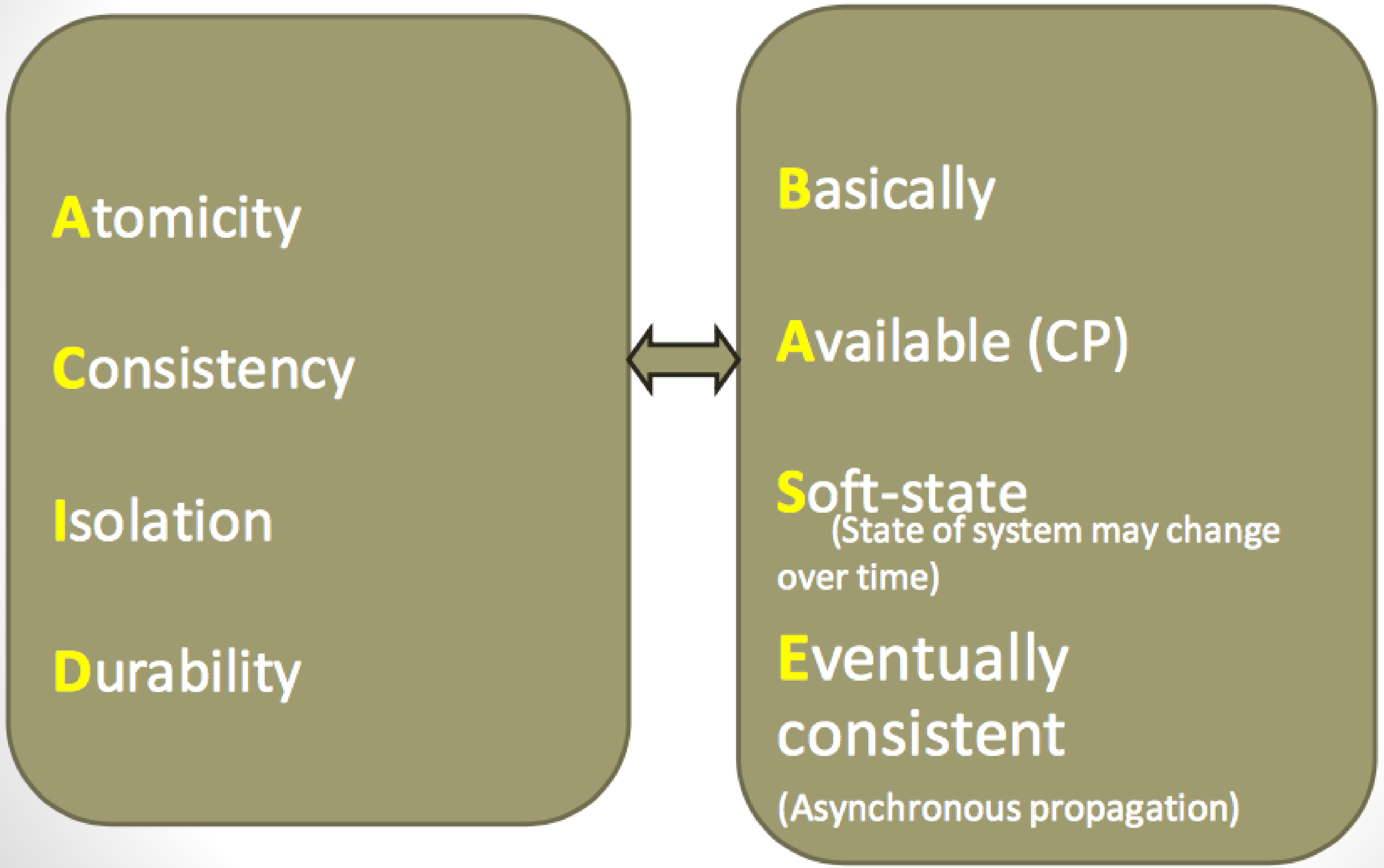
- Support
  - RDBMS vendors provide a high level of support to clients
    - Stellar reputation
  - NoSQL – are open source projects with startups supporting them
    - Reputation not yet established
- Maturity
  - RDMS mature product: means stable and dependable
    - Also means old no longer cutting edge nor interesting
  - NoSQL are still implementing their basic feature set

Drawbacks of NoSQL



- **Administration**
  - RDMS administrator well defined role
  - No SQL's goal: no administrator necessary however NO SQL still requires effort to maintain
- **Lack of Expertise**
  - Whole workforce of trained and seasoned RDMS developers
  - Still recruiting developers to the NoSQL camp
- **Analytics and Business Intelligence**
  - **RDMS designed to address this niche**
  - NoSQL designed to meet the needs of an Web 2.0 application - not designed for ad hoc query of the data
    - Tools are being developed to address this need

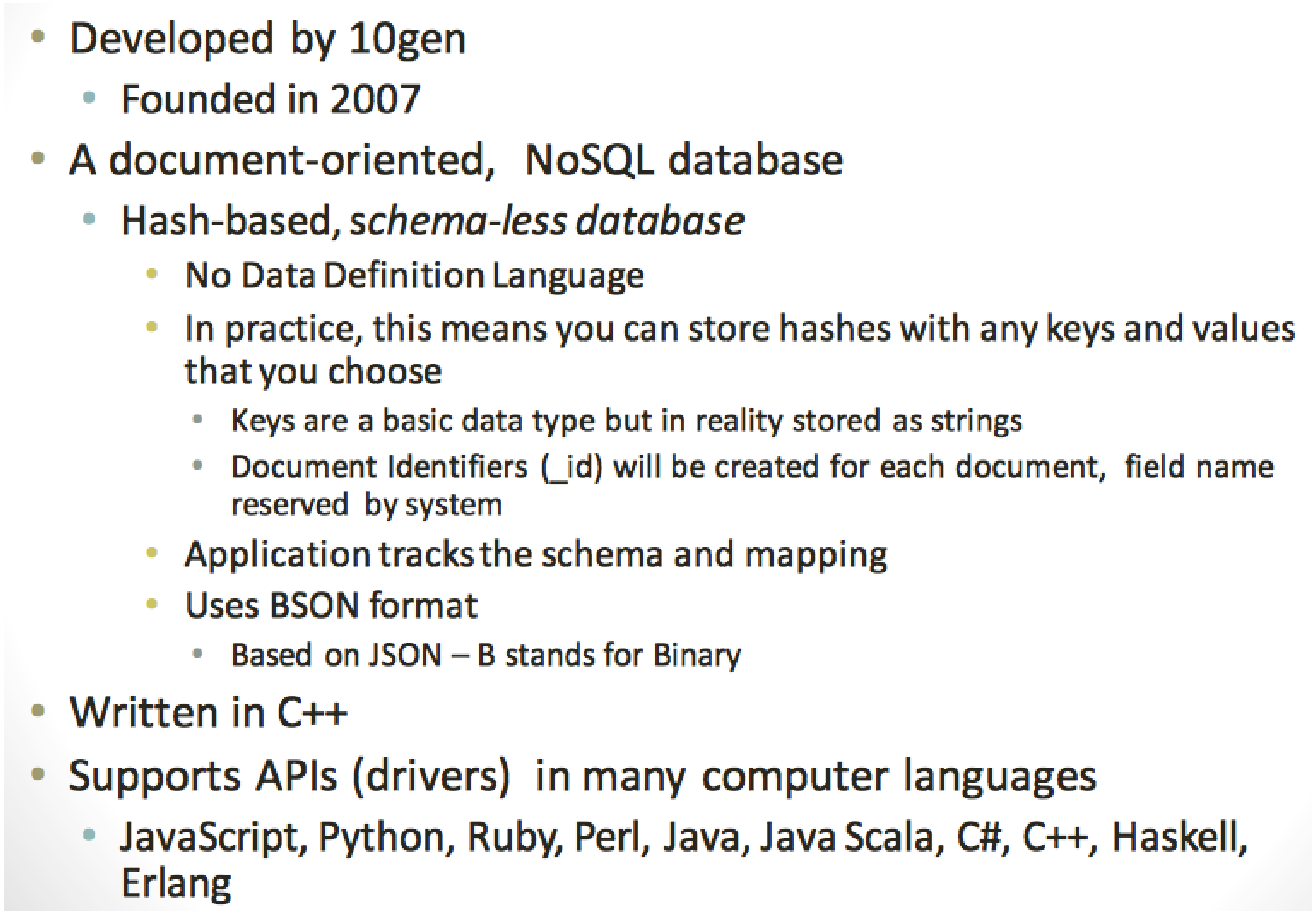
ACID or BASE



Pritchett, D.: BASE: An Acid Alternative (queue.acm.org/detail.cfm?id=1394128)

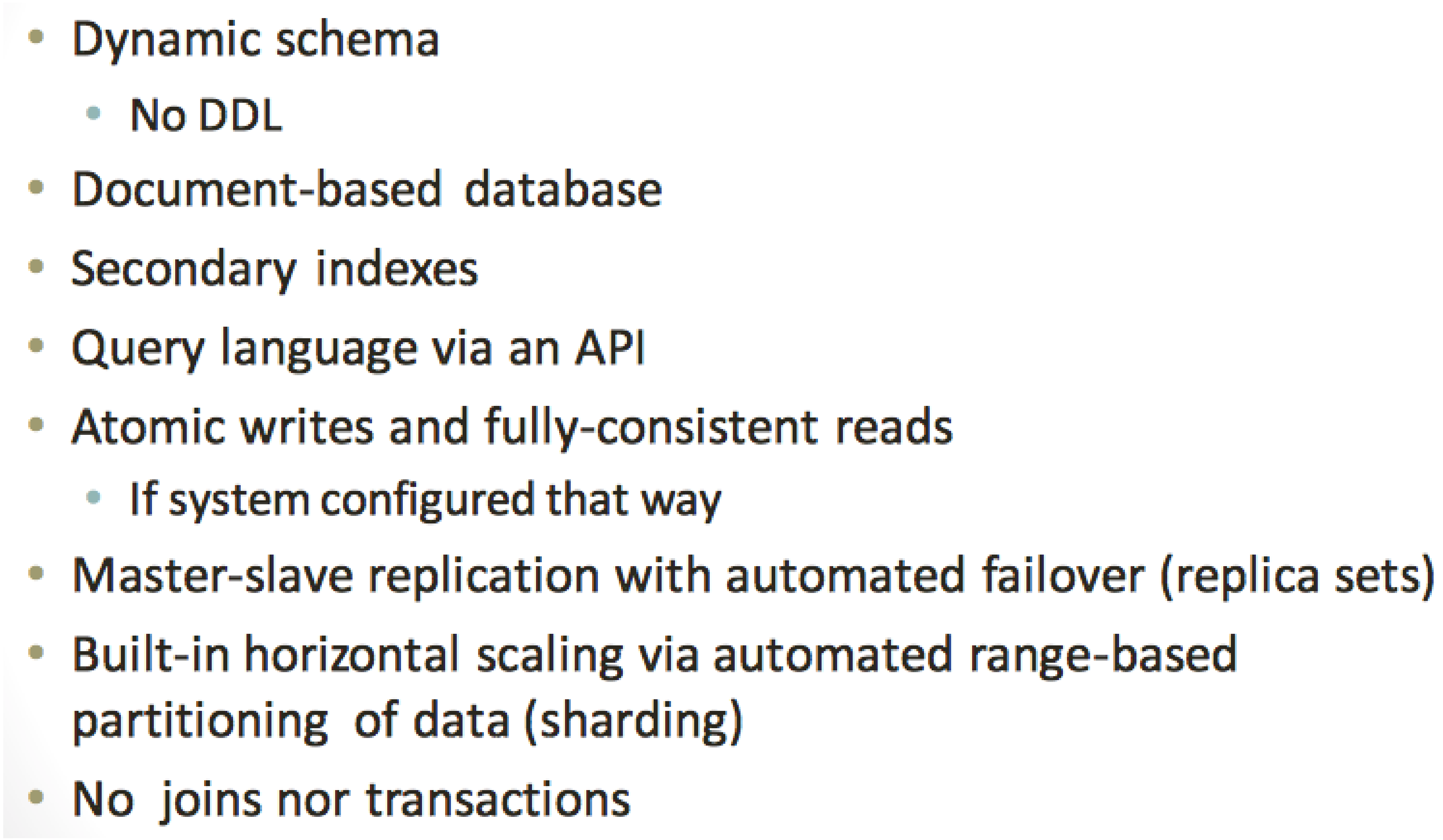
4. MongoDB

What is MongoDB?



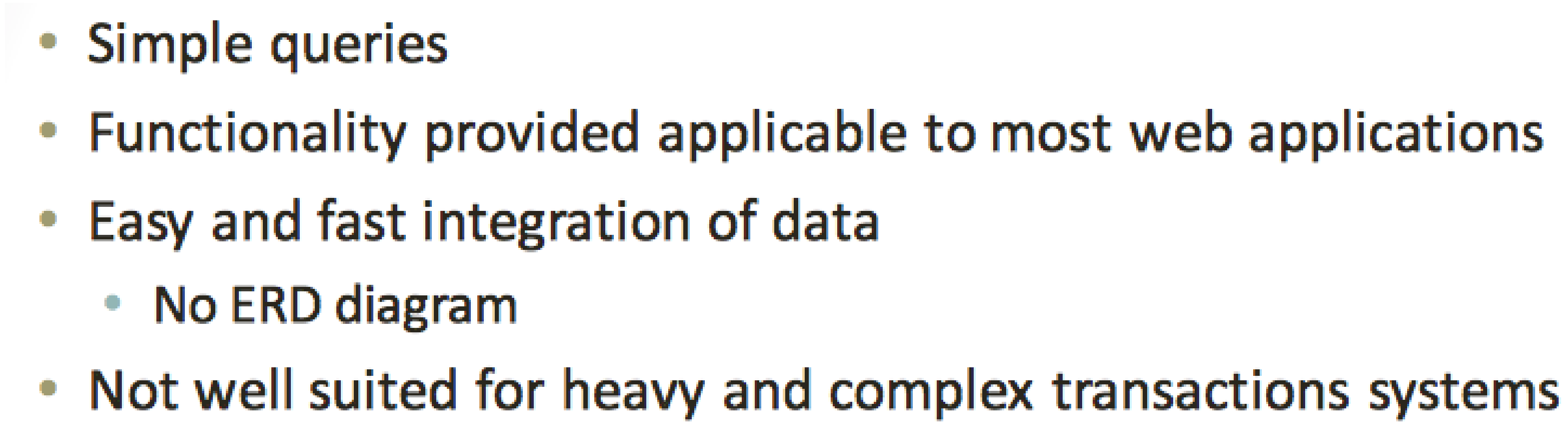
- Developed by 10gen
  - Founded in 2007
- A document-oriented, NoSQL database
  - Hash-based, *schema-less database*
    - No Data Definition Language
    - In practice, this means you can store hashes with any keys and values that you choose
      - Keys are a basic data type but in reality stored 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
- Written in C++
- Supports APIs (drivers) in many computer languages
  - JavaScript, Python, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang

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?

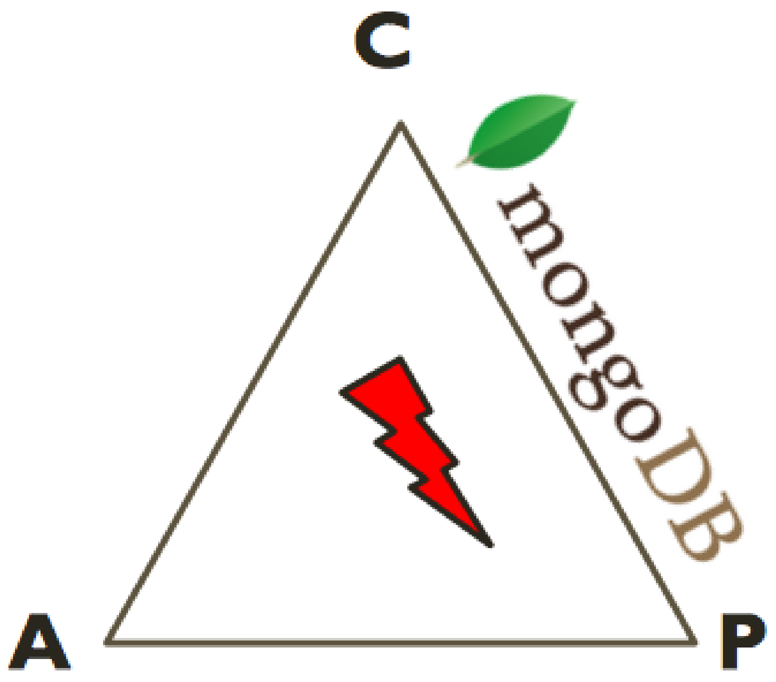


- Simple queries
- Functionality provided applicable to most web applications
- Easy and fast integration of data
  - No ERD diagram
- Not well suited for heavy and complex transactions systems

MongoDB: CAP approach

Focus on Consistency and Partition tolerance

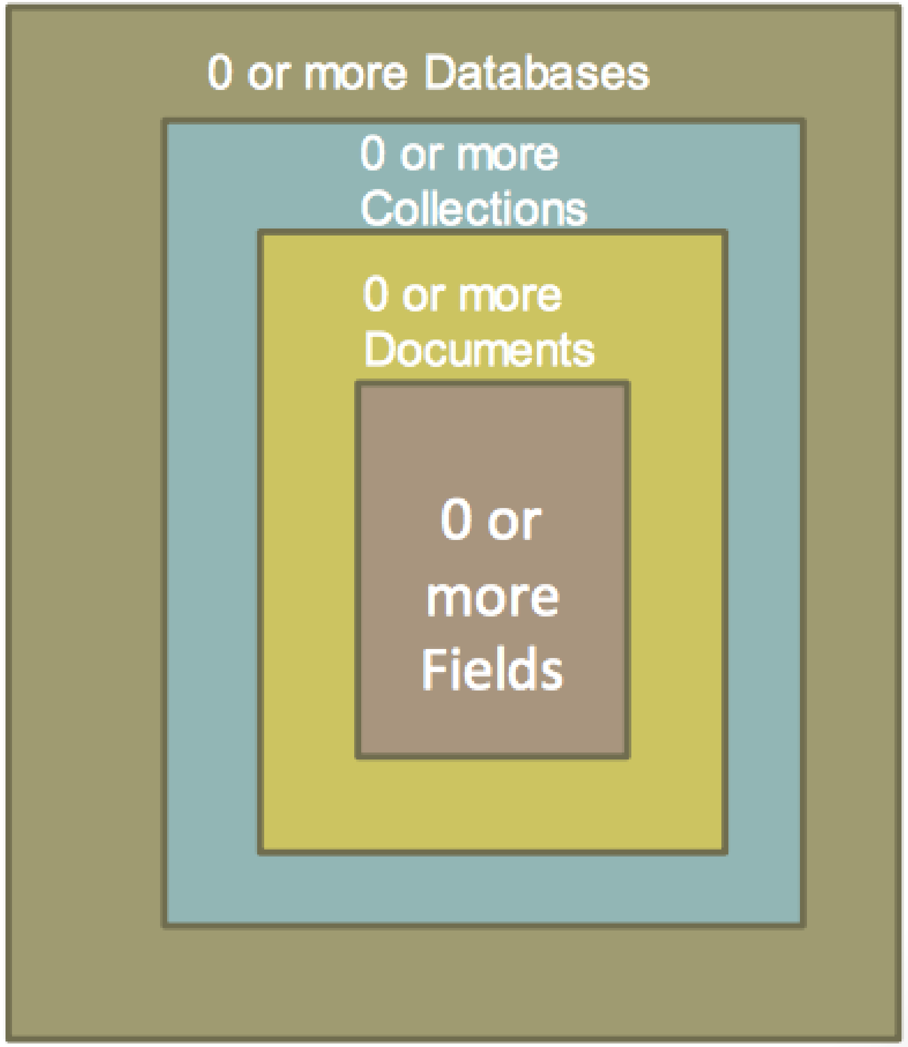
- **Consistency**
  - all replicas contain the same version of the data
- **Availability**
  - system remains operational on failing nodes
- **Partition tolarence**
  - multiple entry points
  - system remains operational on system split



CAP Theorem:
satisfying all three at the same time is impossible

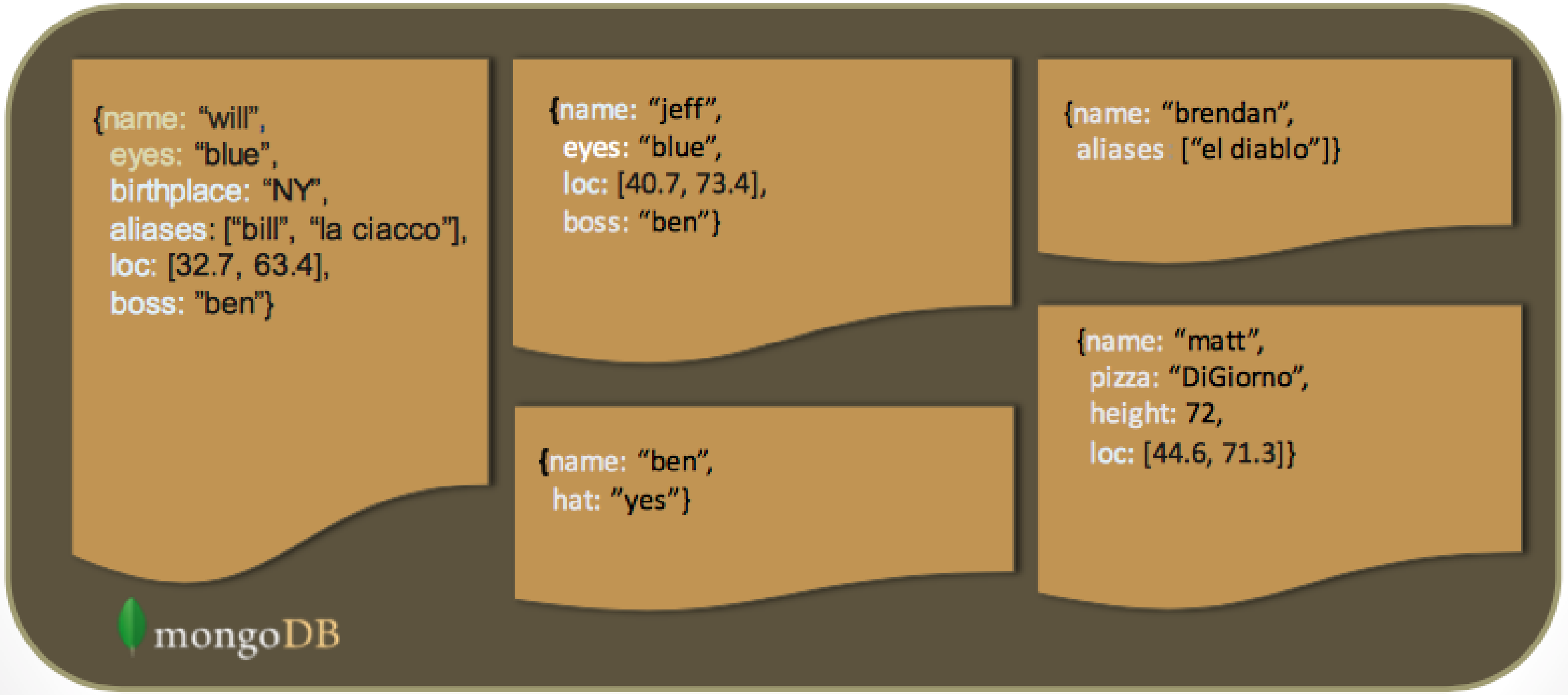
MongoDB Data model: Hierarchical Objects

- 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.



Schema Free

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



```
{name: "will",
  eyes: "blue",
  birthplace: "NY",
  aliases: ["bill", "la ciacco"],
  loc: [32.7, 63.4],
  boss: "ben"}
```

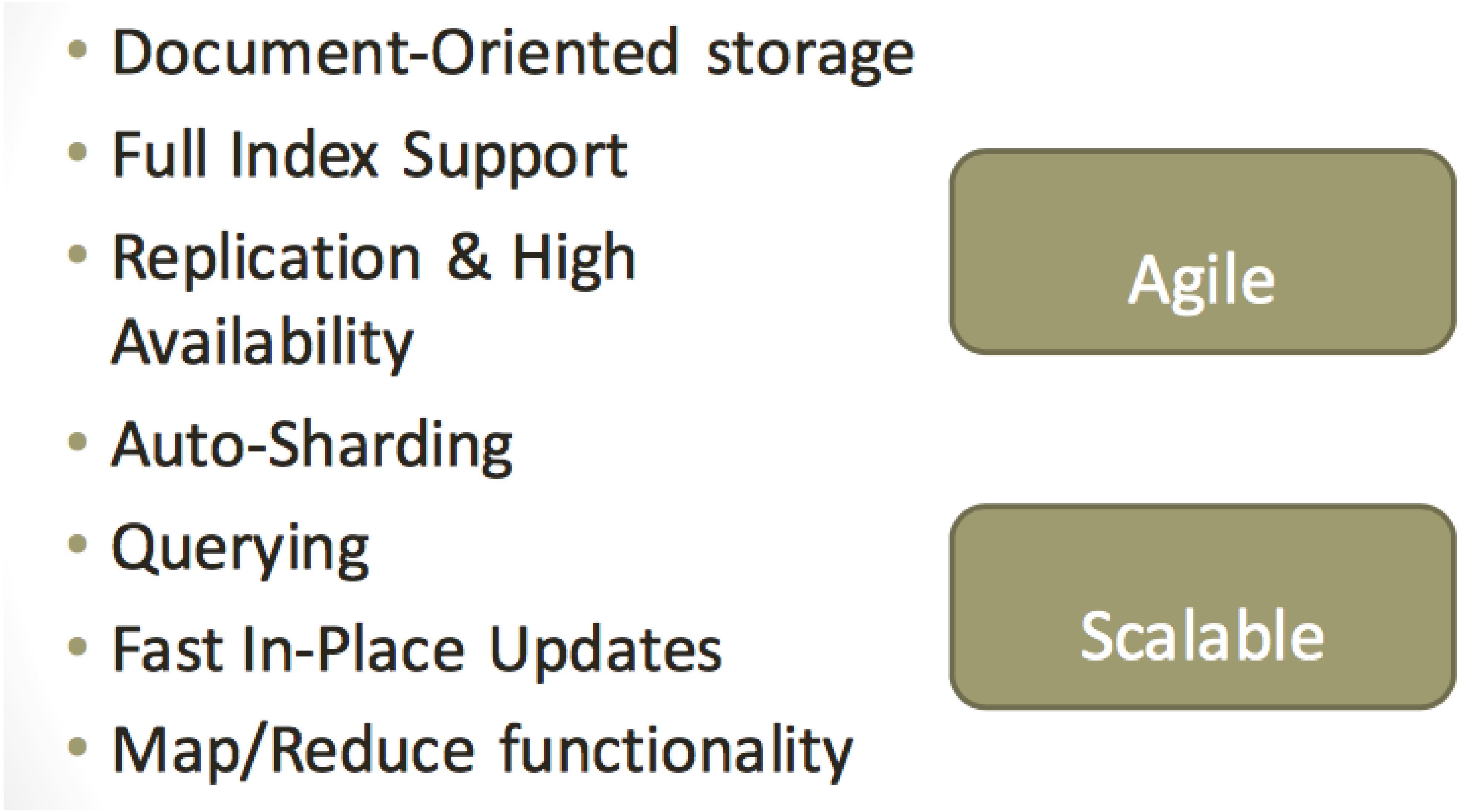
```
{name: "jeff",
  eyes: "blue",
  loc: [40.7, 73.4],
  boss: "ben"}
```

```
{name: "brendan",
  aliases: ["el diablo"]}
```

```
{name: "ben",
  hat: "yes"}
```

```
{name: "matt",
  pizza: "DiGiorno",
  height: 72,
  loc: [44.6, 71.3]}
```

MongoDB Features

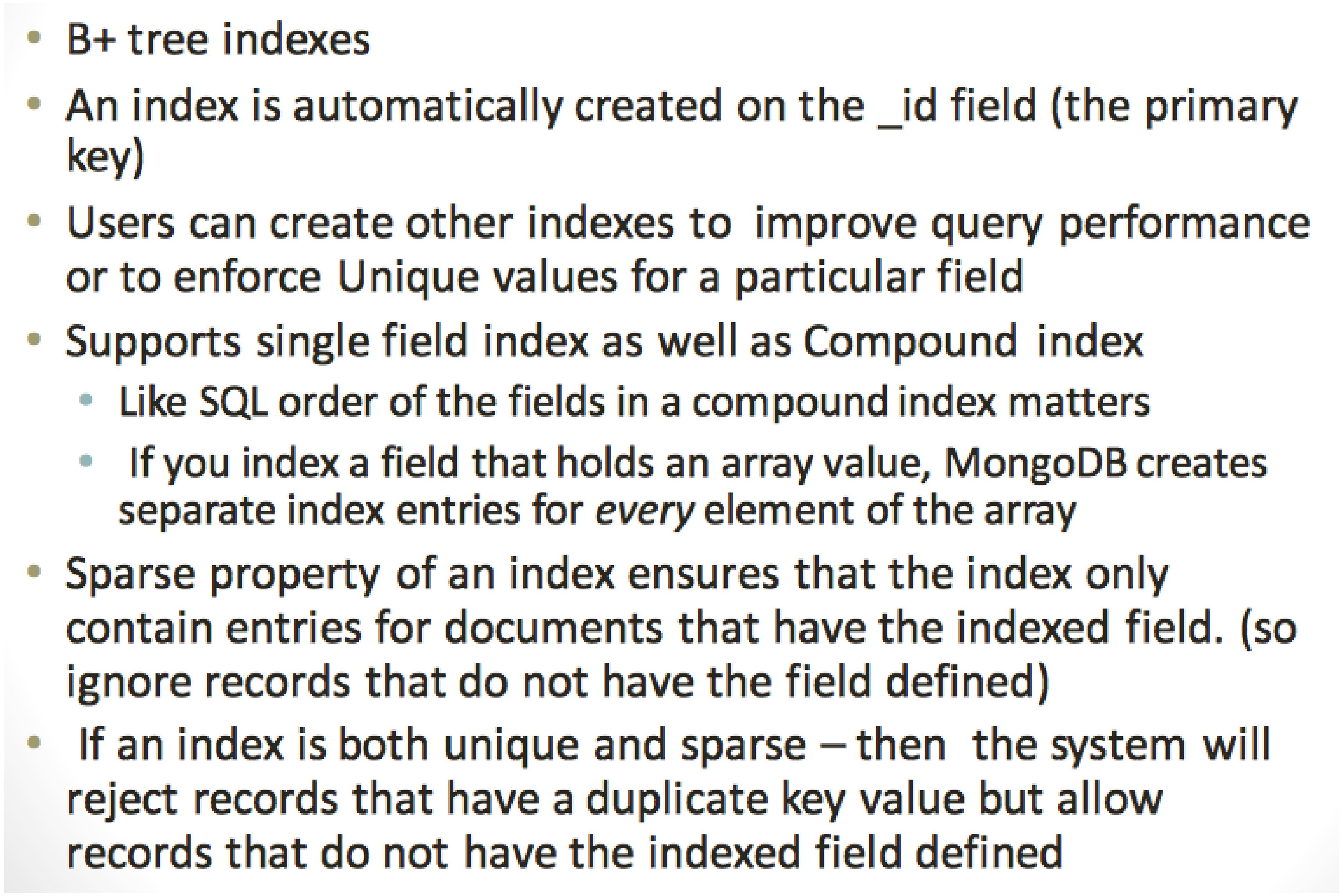


- Document-Oriented storage
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Querying
- Fast In-Place Updates
- Map/Reduce functionality

Agile

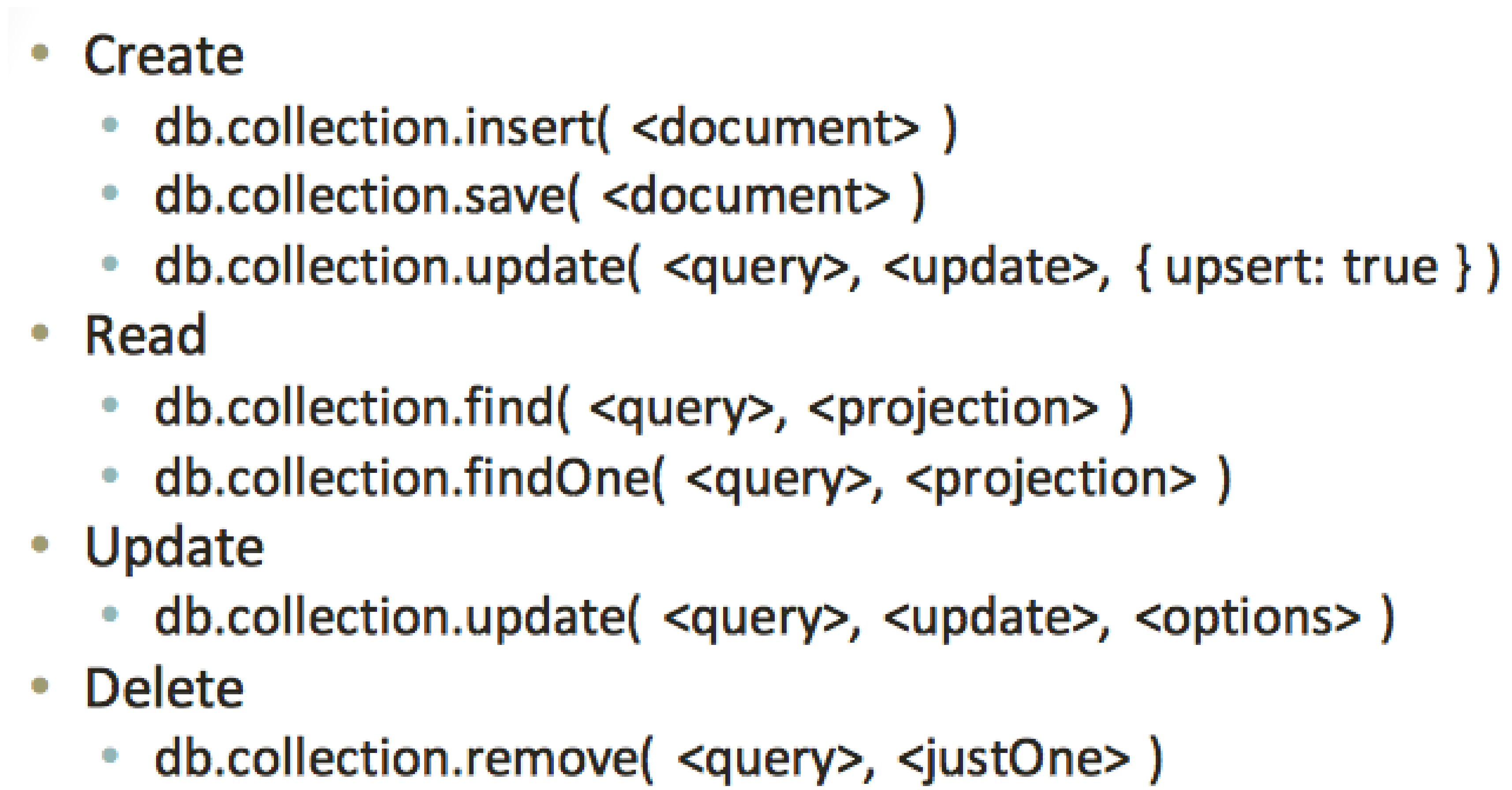
Scalable

Index Functionality



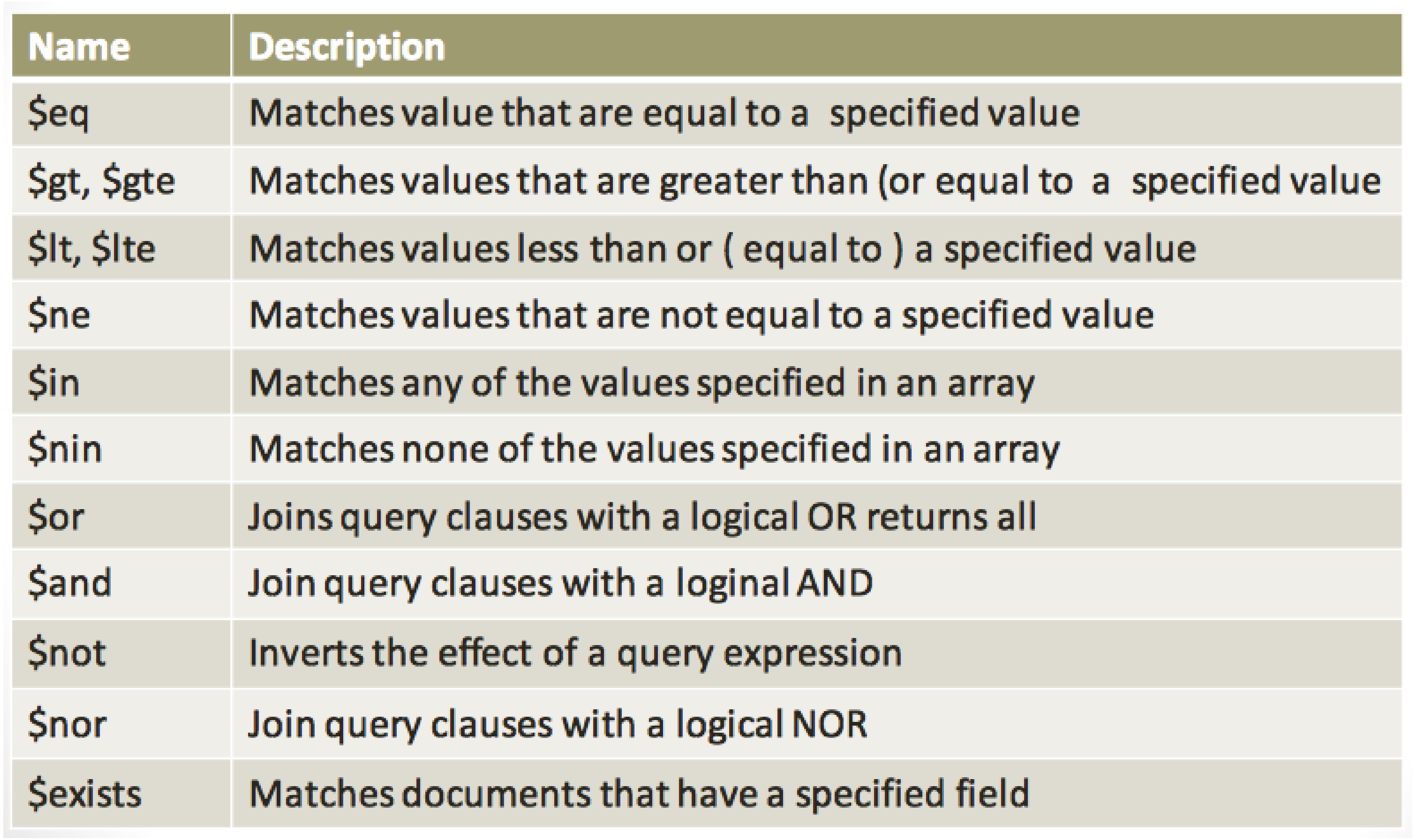
- B+ tree indexes
- An index is automatically created on the _id field (the primary key)
- Users can create other indexes to improve query performance or to enforce Unique values for a particular field
- Supports single field index as well as Compound index
  - Like SQL order of the fields in a compound index matters
  - If you index a field that holds an array value, MongoDB creates separate index entries for *every* element of the array
- Sparse property of an index ensures that the index only contain entries for documents that have the indexed field. (so ignore records that do not have the field defined)
- If an index is both unique and sparse – then the system will reject records that have a duplicate key value but allow records that do not have the indexed field defined

CRUD operations



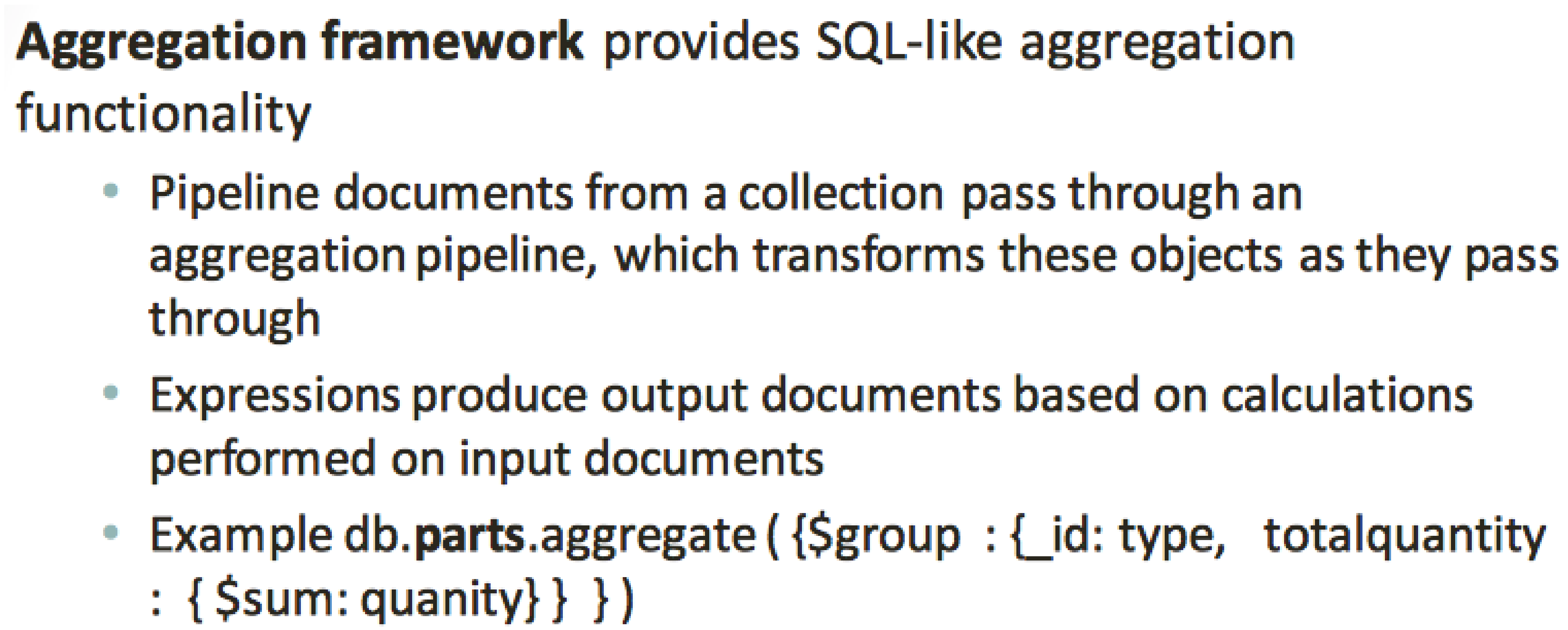
- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

Query operations



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

Aggregated functionality



Aggregation framework provides SQL-like aggregation functionality

- Pipeline documents from a collection pass through an aggregation pipeline, which transforms these objects as they pass through
- Expressions produce output documents based on calculations performed on input documents
- Example db.**parts**.aggregate ({$group : {_id: type, totalquantity : { $sum: quanity} } })

Map reduce functionality

- Performs complex aggregator functions given a collection of keys, value pairs
- Must provide at least a map function, reduction function and a name of the result set
- db.collection.mapReduce(<mapfunction>, <reducefunction>, { out: <collection>, query: <document>, sort: <document>, limit: <number>, finalize: <function>, scope: <document>, jsMode: <boolean>, verbose: <boolean> })
- More description of map reduce next lecture

Summary

- NoSQL built to address a distributed database system
  - Sharding
  - Replica sets of data
- CAP Theorem: consistency, availability and partition tolerant
- MongoDB
  - Document oriented data, schema-less database, supports secondary indexes, provides a query language, consistent reads on primary sets
  - Lacks transactions, joins