# L2. NoSQL[1]

## S1. Principles

Dr A Boronat

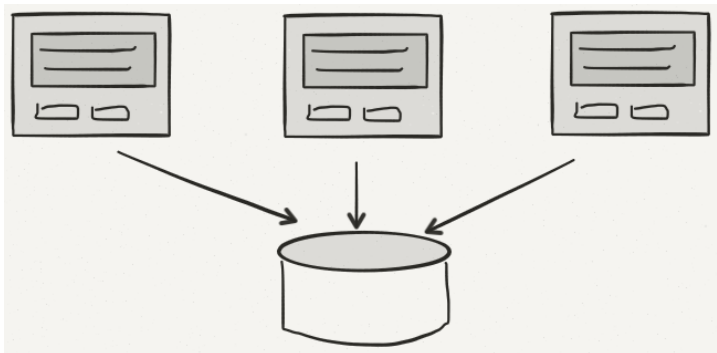
[1] Materials based on [1, ch 1-3]

# Table of Contents

# Up to now...relational databases

- Standard model (relational) and query language (SQL): different vendors' SQL dialects are similar
  - record: set of name-value pairs, represented as a row
  - relation: set of records, represented as a table (with columns and rows)
- ACID transactions: Atomicity, Consistency, Isolation, Durability
  - Atomicity: all or nothing
  - Consistency: any transaction will bring the database from one valid state to another
  - Isolation: the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially
  - Durability: the results need to be stored permanently even in the event of app crashes

- Used as integration mechanism between applications
  - multiple apps store their data in a single db
  - views help defining domain-oriented queries



# Do they cover all current needs?

- Simplicity of data model
  - a relational tuple cannot contain any structure
  - in-memory data structures can be much richer ☹
- Impedance mismatch
  - mapping data between in-memory data structures and a relational db
  - ORM frameworks (Hibernate, JPA-like) help in this task
  - however, once we forget the db structure, our query performance may worsen dramatically ☹

# Do they cover all current needs?

- Integration databases are complex
  - each app is developed by a different team and it may only use part of the database
  - someone needs to ensure the consistency of the whole database ☹
- Big data and traffic
  - scaling up (supercomputer) is expensive
  - scaling out (cluster) may be reasonably affordable
    - cheaper (and also less reliable) machines working together
    - relational databases not designed for clusters ☹

# NoSQL: a new neighbourhood

- Schemaless: more flexibility
  - no need to map structured data
  - rapid development
- Shift from integration dbs to application dbs
  - SOA and micro-services as integration paradigm
  - interoperability concerns at the interface of the application
- Cluster-oriented
  - load balancing
  - ability to trade off traditional consistency for other useful properties (e.g. availability)
- Do not use the relational model
- Tend to be open source

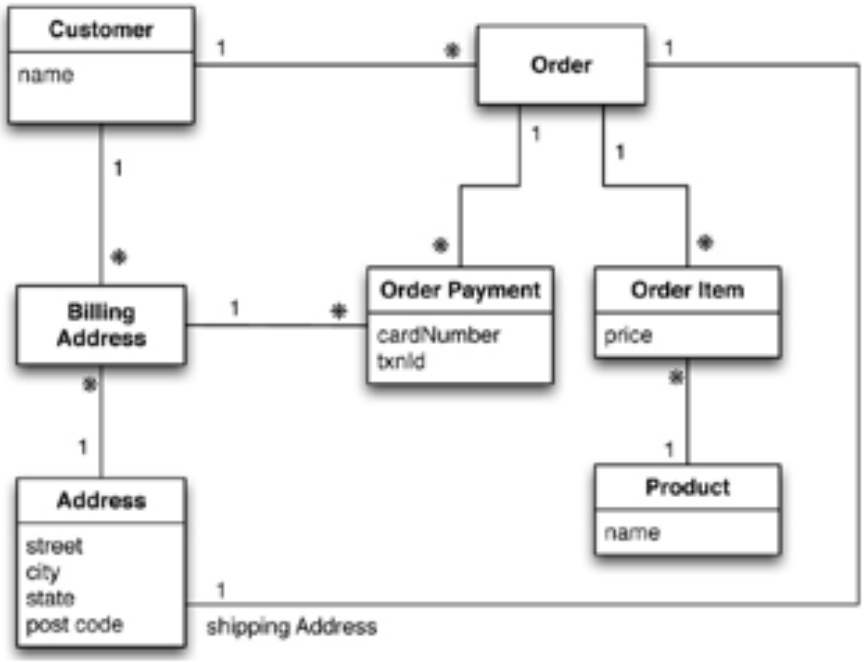
# Reasons

- App dev productivity
  - mapping data between in-memory data structures and a relational db
  - a NoSQL DB simplifies this interaction (less code)
- Large-scale data
  - relational dbs are designed to run on a single machine
  - NoSQL DBs are designed explicitly to run on clusters:
    - better fit for bid data scenarios
    - easier to manage computing loads

# Technology Outline: aggregate-based stores

- Aggregate: collection of data that we interact with as a unit
  - manipulation of multiple aggregates should be dealt with in the app logic
- Form the boundaries for ACID operations with the database
- Aggregation is a physical property, it helps running on a cluster

# (Relational) data model and instance




**Customer**

| Id | Name |
|---|---|
| 1 | Martin |

**Order**

| Id | CustomerId | ShippingAddressId |
|---|---|---|
| 99 | 1 | 77 |

**Product**

| Id | Name |
|---|---|
| 27 | NoSQL Distilled |

**BillingAddress**

| Id | CustomerId | AddressId |
|---|---|---|
| 55 | 1 | 77 |

**OrderItem**

| Id | OrderId | ProductId | Price |
|---|---|---|---|
| 100 | 99 | 27 | 32.45 |

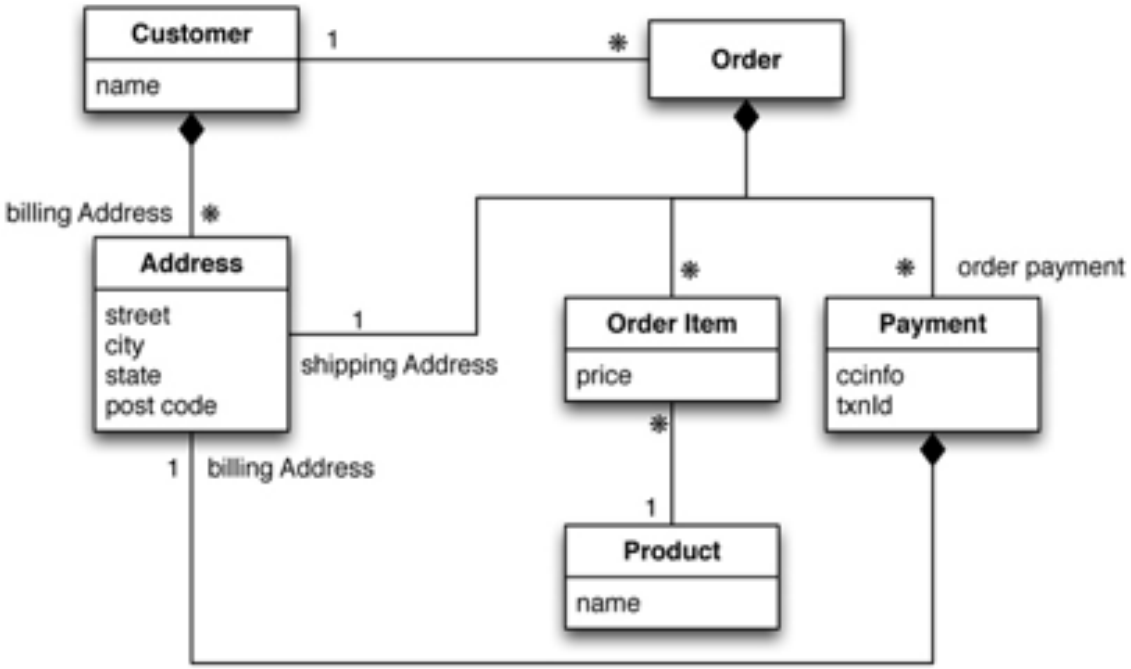
**Address**

| Id | City |
|---|---|
| 77 | Chicago |

**OrderPayment**

| Id | OrderId | CardNumber | BillingAddressId | txnId |
|---|---|---|---|---|
| 33 | 99 | 1000-1000 | 55 | abelif879rft |

# Aggregate data model and instance




```
// in customers
{
"id":1,
"name":"Martin",
"billingAddress":[{"city":"Chicago"}]
}

// in orders
{
"id":99,
"customerId":1,
"orderItems":[
  {
  "productId":27,
  "price": 32.45,
  "productName": "NoSQL Distilled"
    }
  ],
"shippingAddress":[{"city":"Chicago"}]
"orderPayment":[
  {
    "ccinfo":"1000-1000-1000-1000",
    "txnId":"abelif879rft",
    "billingAddress": {"city": "Chicago"}
  }
  ],
}
```

# Technology Outline: aggregate-based stores

- Key-value stores:
  - a db consists of several aggregates, each of which has a key or ID used to look up the data
  - the aggregate is opaque to the db
    Examples: Redis, Riak
- Document stores:
  - query based on the internal structure of the document, which can be a key
  - db is able to see the structure of the aggregate
  - can retrieve part of the aggregate rather than the whole thing
    Examples: CouchDB, MongoDB
- Column-family:
  - divide the aggregate into column families
  - aggregate normally computed using map-reduce algorithm
    Examples: HBase, Apache Cassandra

# Technology Outline: graph databases

- Reject SQL model
  - small records with complex interconnections
  - traversal of links is very efficient
- Suitable when
  - more queries than inserts (analytics)
  - data with complex relationships: social networks, product preferences, eligibility rules
- More likely to run on a single server
- ACID transactions need to cover multiple nodes and edges to cover consistency
- Example: Neo4J
  - allows us to attach Java objects as properties (features) to nodes and edges in a schemaless fashion

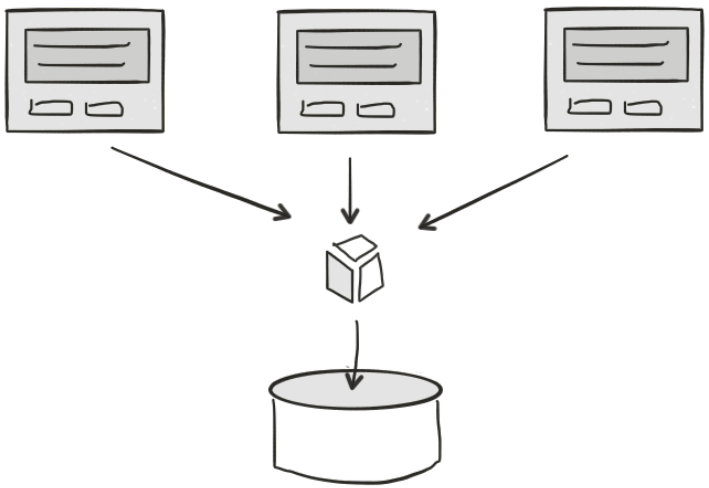
# Working without Schemas

- NoSQL is untyped: advantages
  - flexibility for prototyping within an aggregate
  - nonuniform data: each record has a different set of fields (one row per table)
  - reporting of structured data
- Working with schemas introduces discipline in programming
  - lacking a schema may introduce errors that can only be found at run time
  - the code needs to be inspected to find bugs
- The schema is implicit
  - the schema is shifted to the code
  - db remains ignorant of schema
    - worse performance
    - data integrity cannot be guaranteed
  - migration as cumbersome as in relational dbs in some cases
- Queries that select rows cannot be defined (use indexes instead)

# Integration with NoSQL: approaches

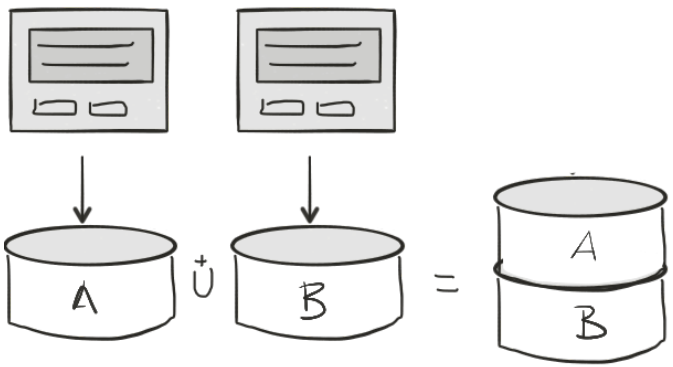
## through a dedicated application

- encapsulate all db interaction within a single app
- integrate it with other apps using web services



## aggregates linked to specific apps

- db is partitioned according to apps



# BIG DATA

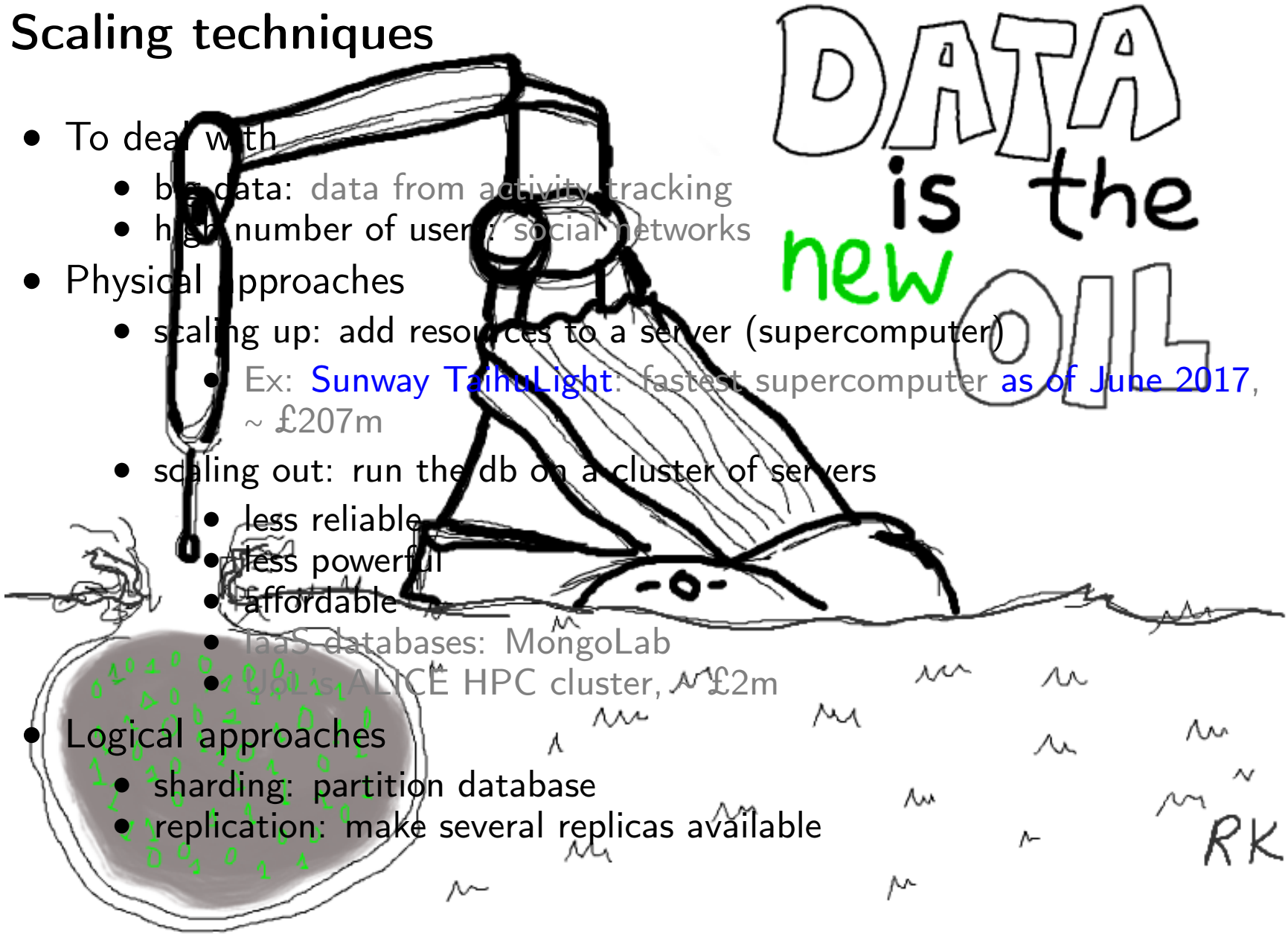
**VOLUME**
DATA SIZE

**VELOCITY**
SPEED OF CHANGE

**VARIETY**
DIFFERENT FORMS OF DATA SOURCES

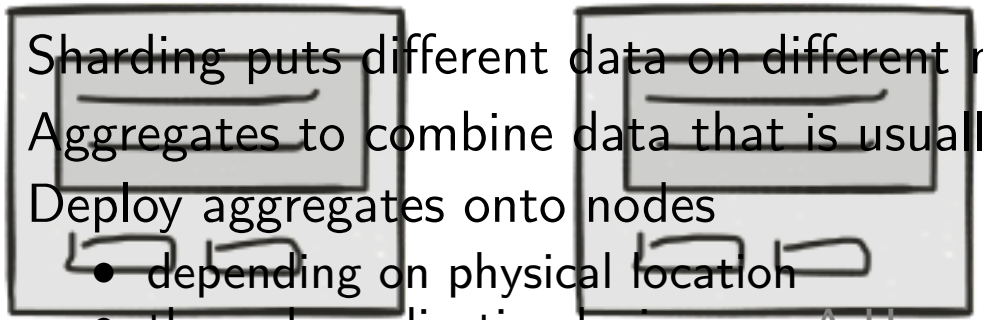
**VERACITY**
UNCERTAINTY OF DATA

# Scaling techniques

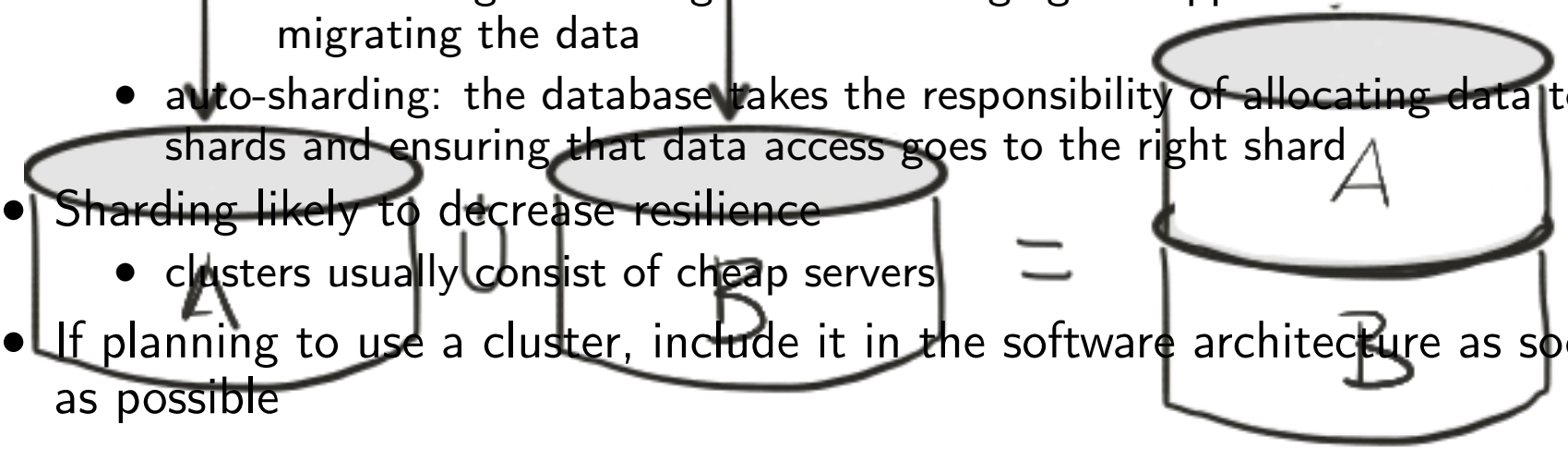



- To deal with
  - big data: data from activity tracking
  - high number of users: social networks
- Physical approaches
  - scaling up: add resources to a server (supercomputer)
    - Ex: Sunway TaihuLight: fastest supercomputer as of June 2017, ~ £207m
  - scaling out: run the db on a cluster of servers
    - less reliable
    - less powerful
    - affordable
    - IaaS databases: MongoLab
    - UoL's ALICE HPC cluster, ~ £2m
- Logical approaches
  - sharding: partition database
  - replication: make several replicas available

# Sharding



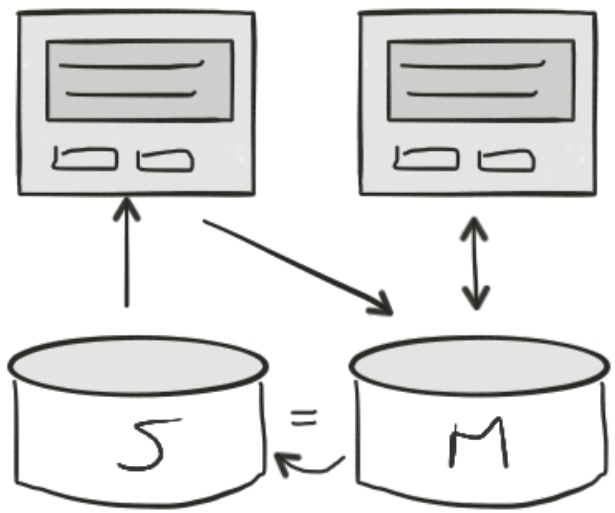
- Sharding puts different data on different nodes
- Aggregates to combine data that is usually accessed together
- Deploy aggregates onto nodes
  - depending on physical location
  - through application logic, e.g., A-H and I-Z
    - rebalancing a sharding involves changing the application code and migrating the data
  - auto-sharding: the database takes the responsibility of allocating data to shards and ensuring that data access goes to the right shard




- Sharding likely to decrease resilience
  - clusters usually consist of cheap servers
- If planning to use a cluster, include it in the software architecture as soon as possible

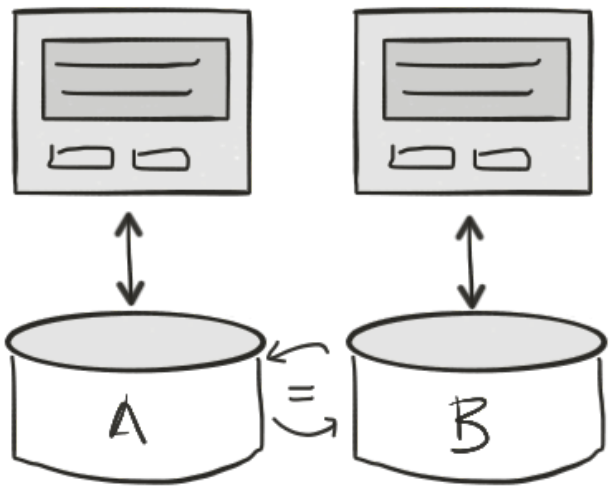
# Replication

- Replicas are deployed onto different nodes
- Master-slave replication
  - master: responsible for processing any updates to data
    - reduces the chance of update (write) conflicts
  - slave: read-only
    - can be used as hot-backup
    - scale out by adding slave nodes
  - advantages
    - balances read traffic
    - read resilience: should the master fail, slave can still handle read requests
  - problems
    - inconsistencies when updates have not been propagated to slaves
    - resilience: the master is a bottleneck and a single point of failure for writes



# Replication

- Peer-to-Peer Replication
  - all the replicas have equal weight and they can all accept writes
  - advantages
    - more resilient (when a node fails the others still work)
  - problems
    - consistency: write-write conflicts (update the same record at the same time)
    - replicas coordinate to ensure we avoid a conflict (more traffic)
    - cope with an inconsistent write by allowing them and merging inconsistent writes



# What to remember

- Welcome to polyglot persistence:
  - Aggregate-oriented databases facilitate the use of clusters
  - Graph databases suitable for complex relationship structures
  - Schemaless databases enable rapid prototyping but there is an implicit schema
- Scaling techniques
  - Sharding distributes different data across multiple servers
  - Replication copies data across multiple servers

# References


Pramod J. Sadalage and Martin Fowler.
*NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*.
Addison-Wesley Professional, 1st edition, 2012.