

Data Engineering

LTAT.02.007

Ass Prof. Riccardo Tommasini

Assistants: **Fabiano Spiga, Mohamed Ragab, Hassan Eldeeb**

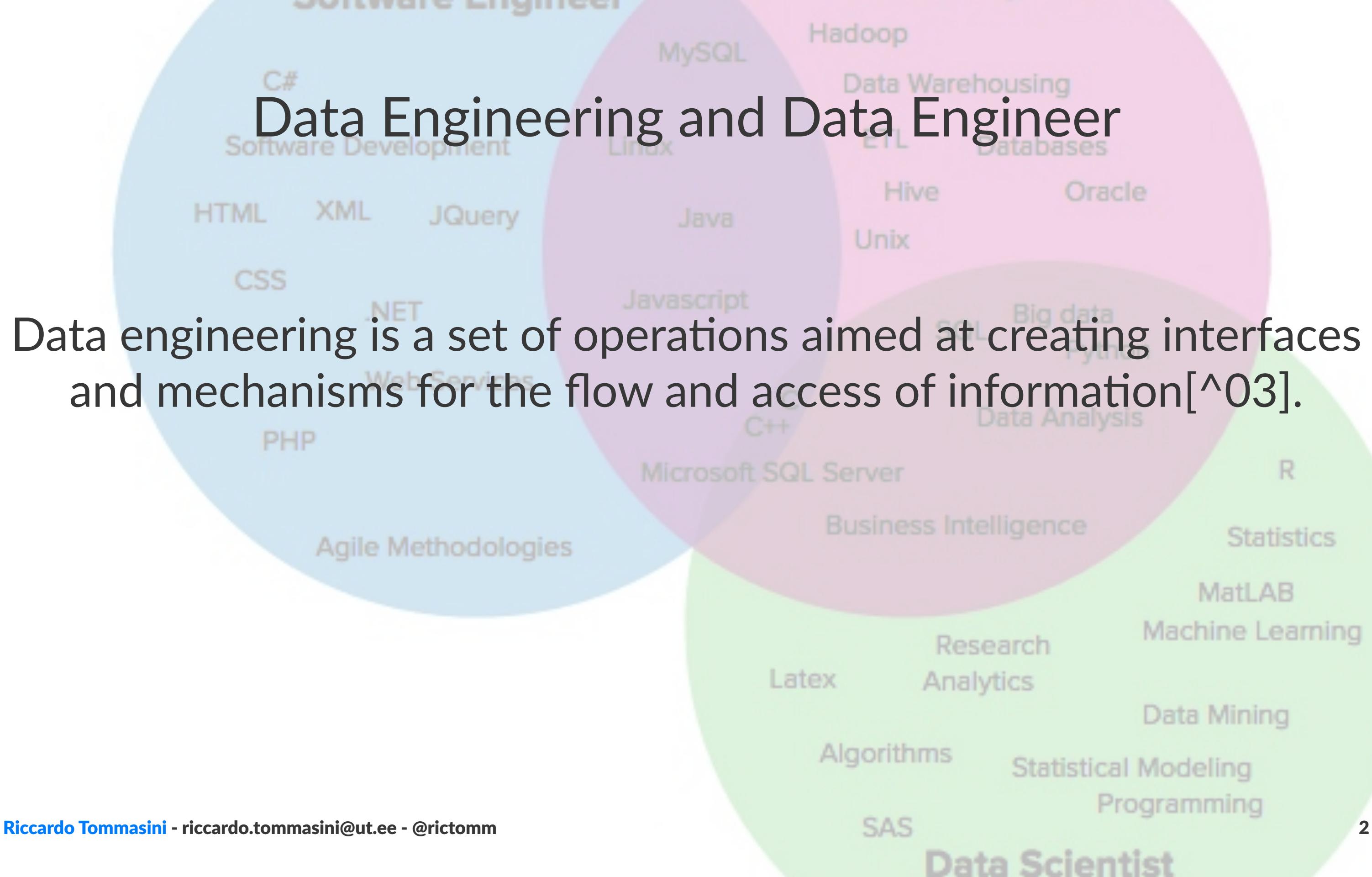


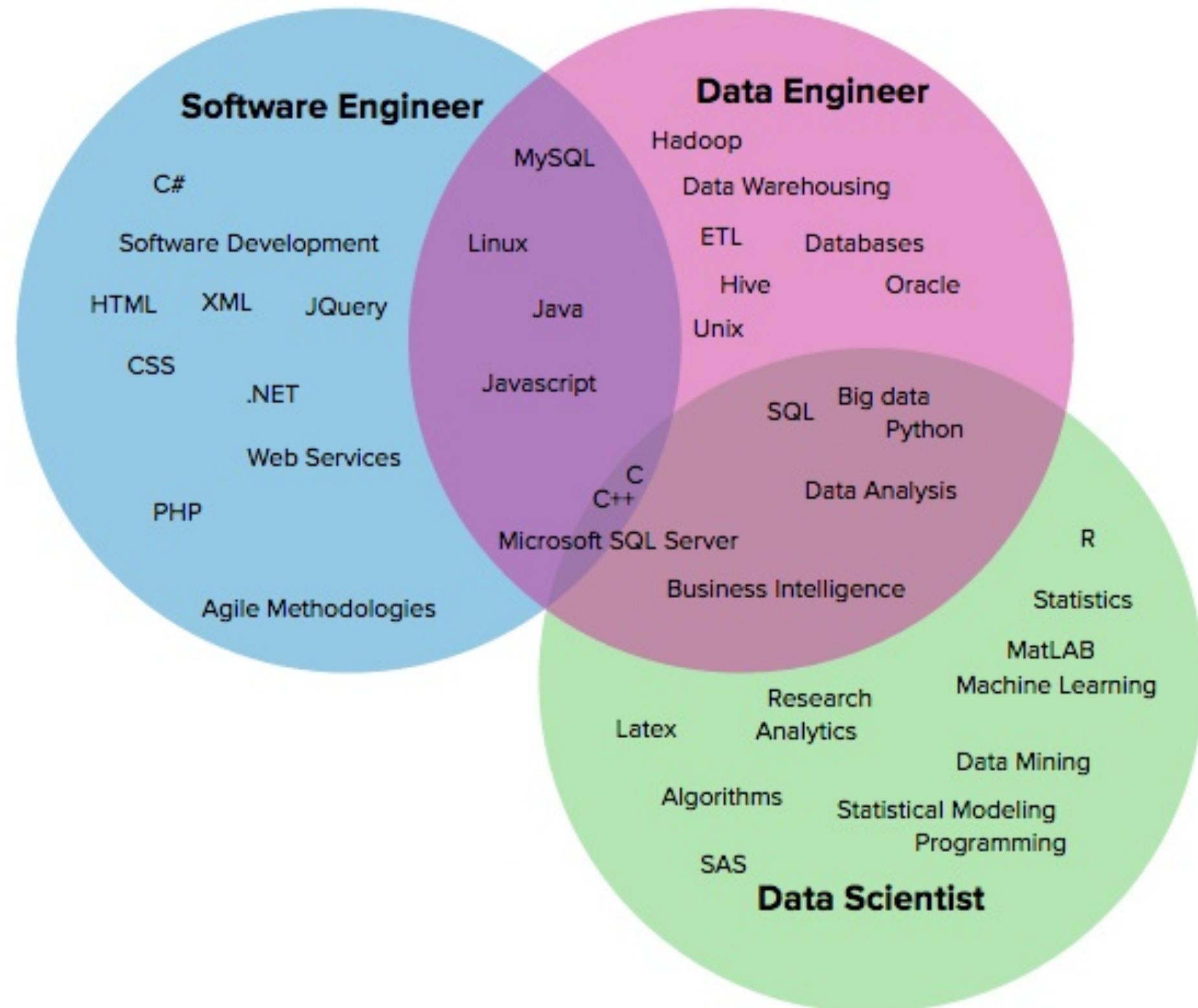
[https://courses.cs.ut.ee/2020/
dataeng](https://courses.cs.ut.ee/2020/dataeng)

Forum

Moodle

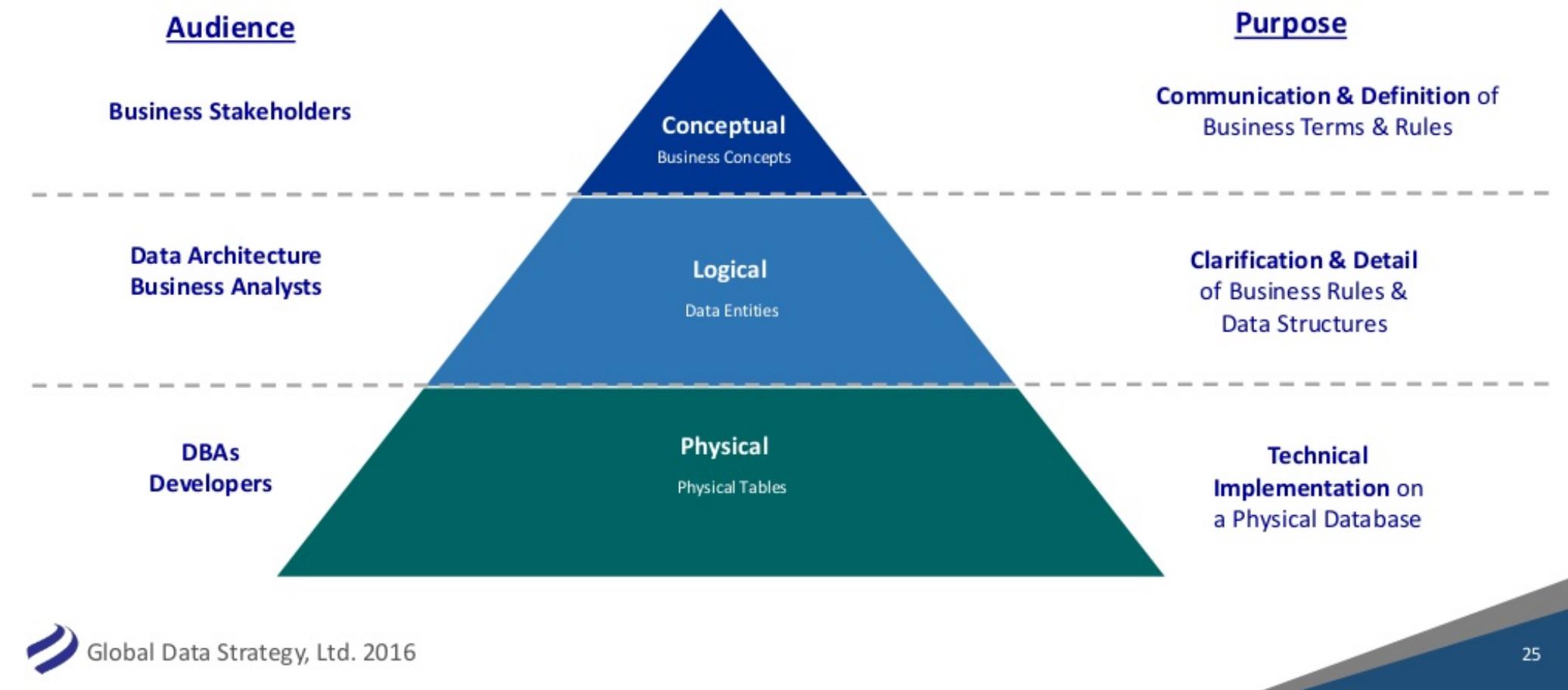






Data Modelling

Levels of Data Modeling



CAP Theorem (Brewer's Theorem)

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

CAP Theorem (Brewer's Theorem)

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- **Consistency**: ???

CAP Theorem (Brewer's Theorem)

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- **Consistency**: ???
- **Availability**: ???

CAP Theorem (Brewer's Theorem)

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- **Consistency**: ???
- **Availability**: ???
- **Partition tolerance**: ???

CAP Theorem (Brewer's Theorem)

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- **Consistency**: all nodes see the same data at the same time
- **Availability**: ???
- **Partition tolerance**: ???

CAP Theorem (Brewer's Theorem)

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- **Consistency**: all nodes see the same data at the same time
- **Availability**: Node failures do not prevent other survivors from continuing to operate (a guarantee that every request receives a response whether it succeeded or failed)
- **Partition tolerance**: ???

CAP Theorem (Brewer's Theorem)

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- **Consistency**: all nodes see the same data at the same time
- **Availability**: Node failures do not prevent other survivors from continuing to operate (a guarantee that every request receives a response whether it succeeded or failed)
- **Partition tolerance**: the system continues to operate despite arbitrary partitioning due to network failures (e.g., message loss)

Data Replication

Replication means keeping a copy of the same data on multiple machines that are connected via a network

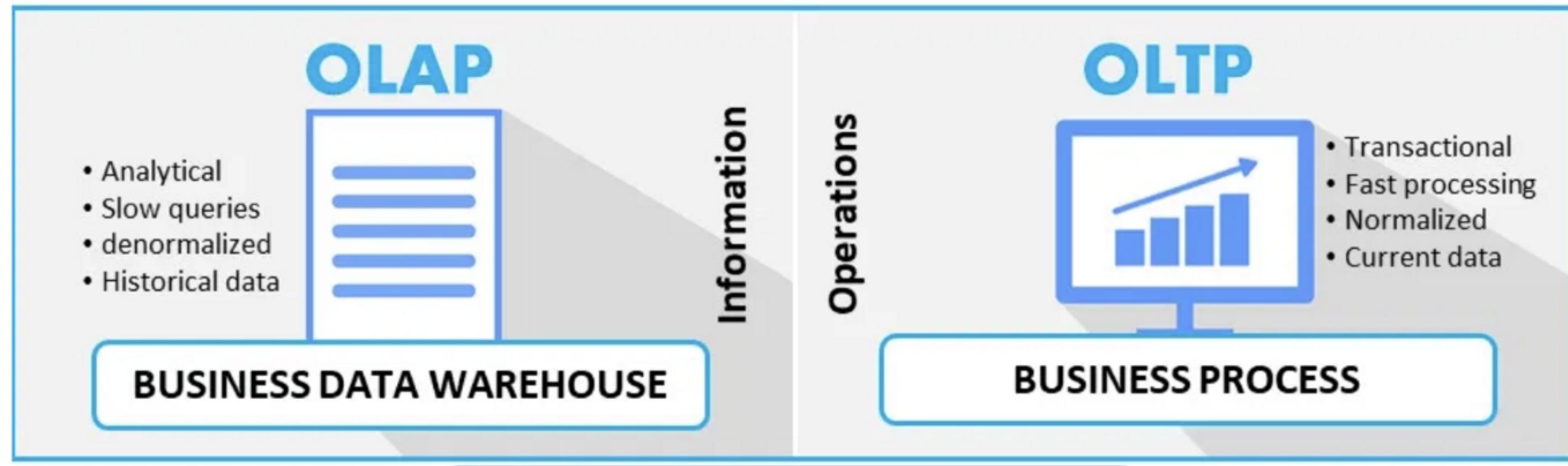


Data Partitioning (Sharding)

breaking a large database down into smaller ones

We Talked about Workloads

OLAP Vs OLTP



Data Modelling for Databases

[Relational Algebra]

is a procedural language consisting of a six basic operations that take one or two relations as input and produce a new relation as their result:

[Relational Algebra]

is a procedural language consisting of a six basic operations that take one or two relations as input and produce a new relation as their result:

- select: σ

[Relational Algebra]

is a procedural language consisting of a six basic operations that take one or two relations as input and produce a new relation as their result:

- select: σ
- project: Π

[Relational Algebra]

is a procedural language consisting of a six basic operations that take one or two relations as input and produce a new relation as their result:

- select: σ
- project: Π
- union: \cup

[Relational Algebra]

is a procedural language consisting of a six basic operations that take one or two relations as input and produce a new relation as their result:

- select: σ
- project: Π
- union: \cup
- set difference: $-$

[Relational Algebra]

is a procedural language consisting of a six basic operations that take one or two relations as input and produce a new relation as their result:

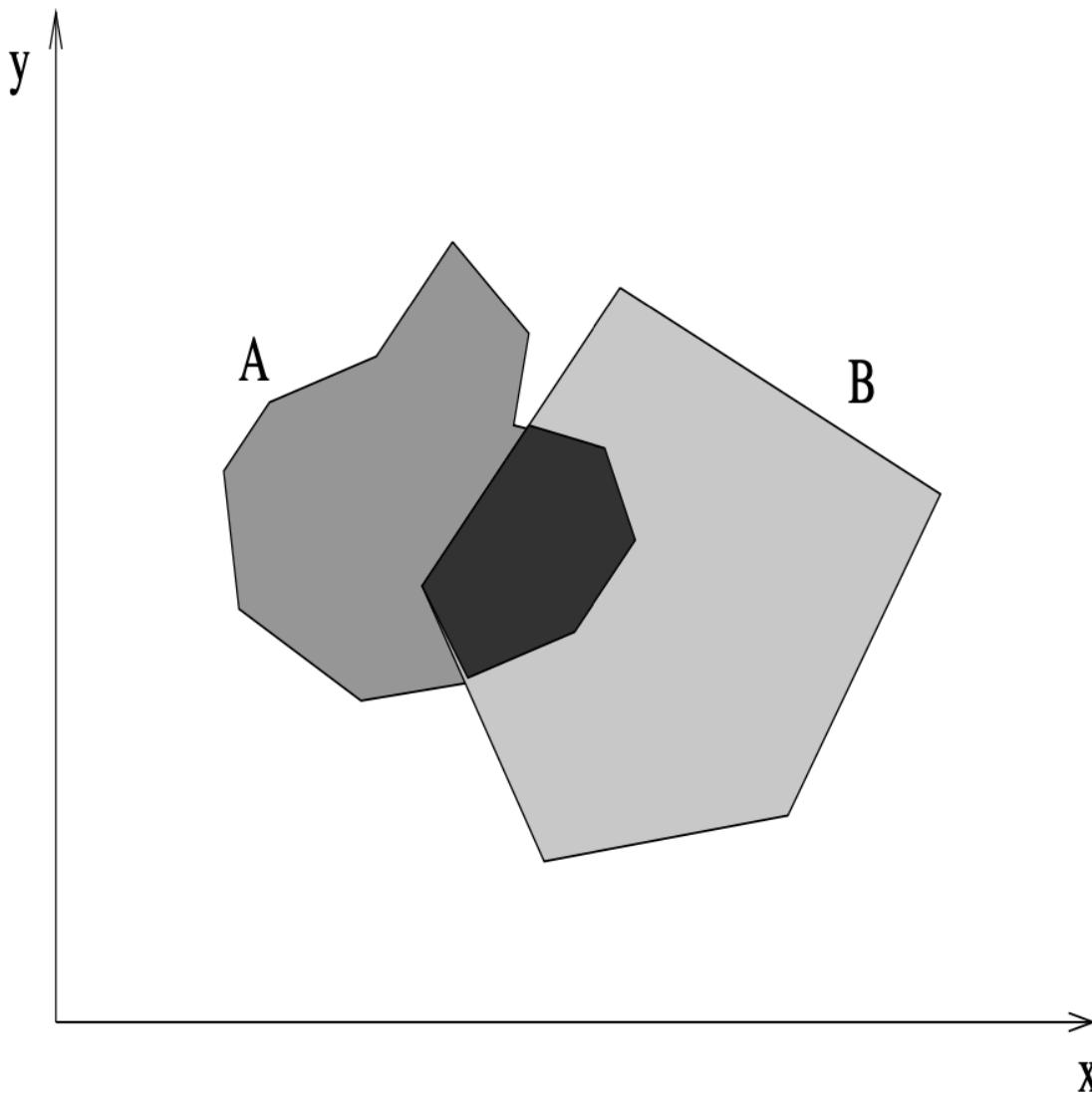
- select: σ
- project: Π
- union: \cup
- set difference: $-$
- Cartesian product: \times

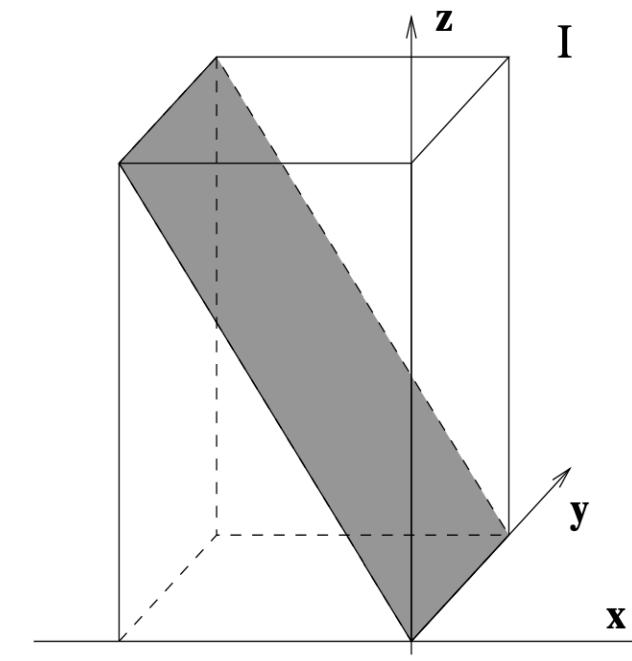
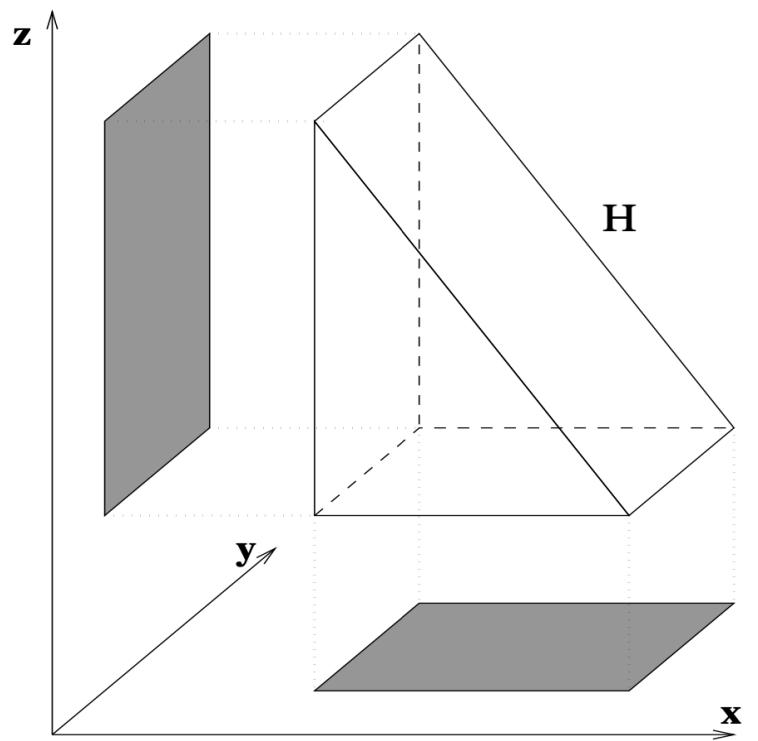
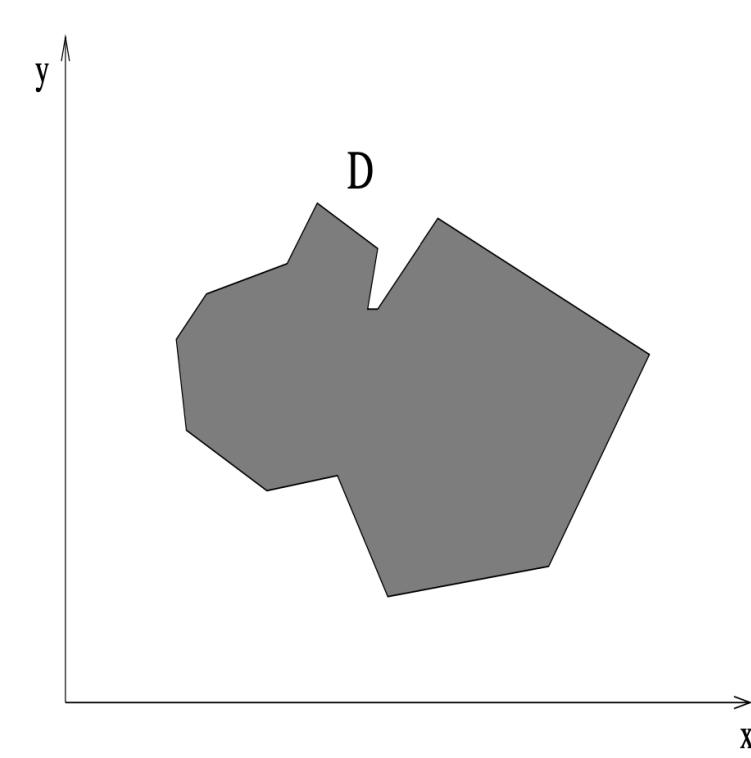
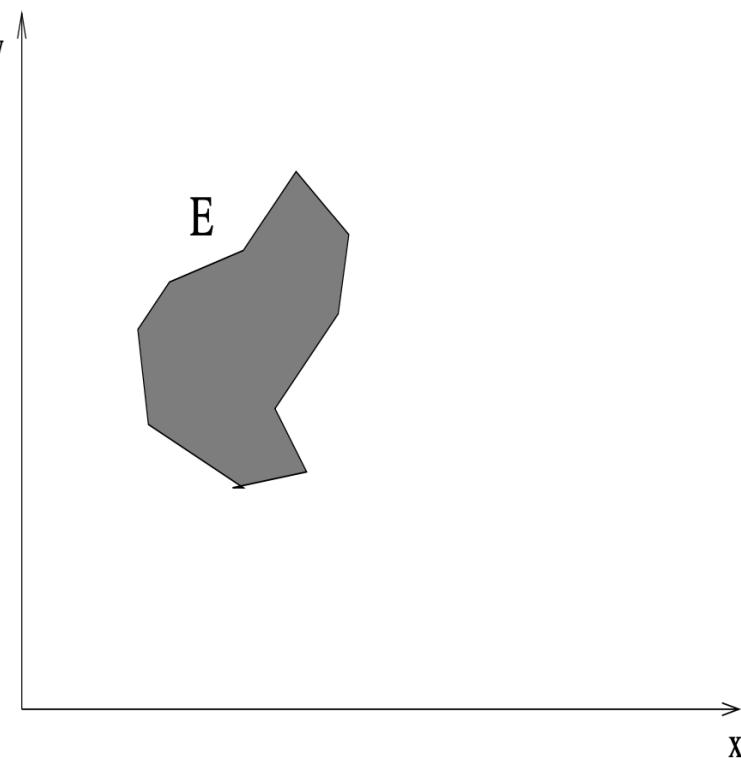
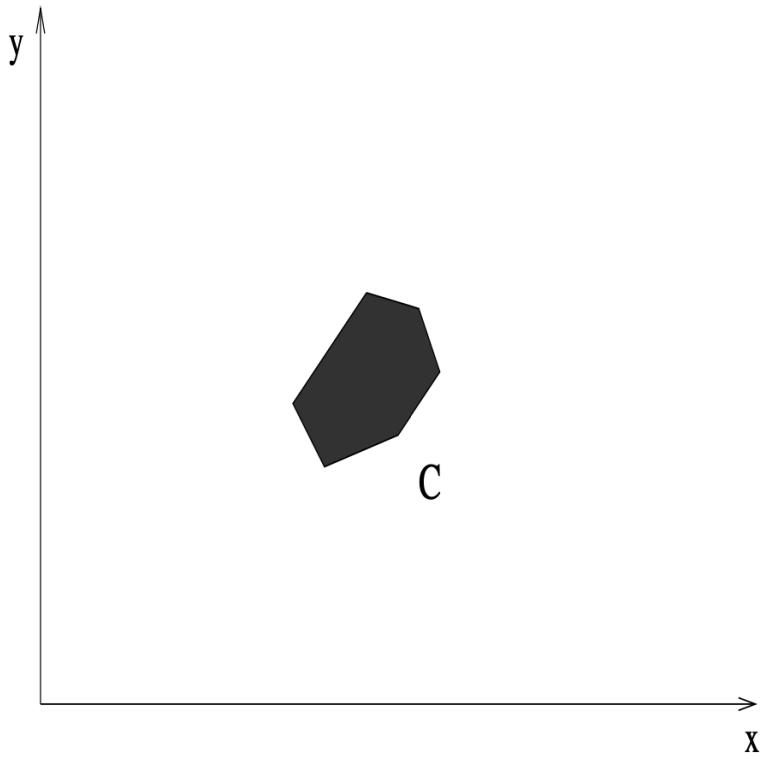
[Relational Algebra]

is a procedural language consisting of a six basic operations that take one or two relations as input and produce a new relation as their result:

- select: σ
- project: Π
- union: \cup
- set difference: $-$
- Cartesian product: \times
- rename: ρ

[[Relational Algebra]] Visualized





Refresh on ACID Properties

Refresh on ACID Properties

- ACID, which stands for Atomicity, Consistency, Isolation, and Durability[^11]

Refresh on ACID Properties

- ACID, which stands for Atomicity, Consistency, Isolation, and Durability[^11]
- **Atomicity** refers to something that cannot be broken down into smaller parts.

Refresh on ACID Properties

- ACID, which stands for Atomicity, Consistency, Isolation, and Durability[^11]
- **Atomicity** refers to something that cannot be broken down into smaller parts.
 - It is not about concurrency (which comes with the I)

Refresh on ACID Properties

- ACID, which stands for Atomicity, Consistency, Isolation, and Durability[^11]
- **Atomicity** refers to something that cannot be broken down into smaller parts.
 - It is not about concurrency (which comes with the I)
- **Consistency** (overused term), that here relates to the data *invariants* (integrity would be a better term IMHO)

Refresh on ACID Properties

- ACID, which stands for Atomicity, Consistency, Isolation, and Durability[^11]
- **Atomicity** refers to something that cannot be broken down into smaller parts.
 - It is not about concurrency (which comes with the I)
- **Consistency** (overused term), that here relates to the data *invariants* (integrity would be a better term IMHO)
- **Isolation** means that concurrently executing transactions are isolated from each other.

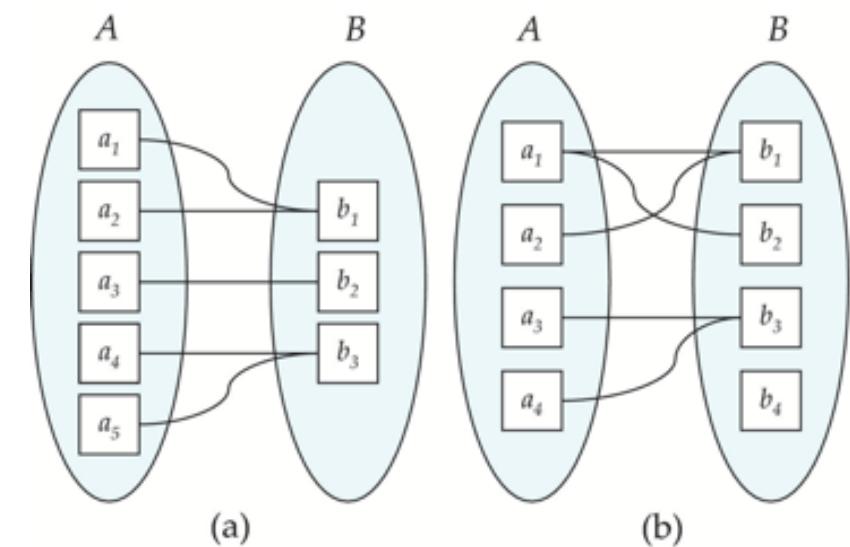
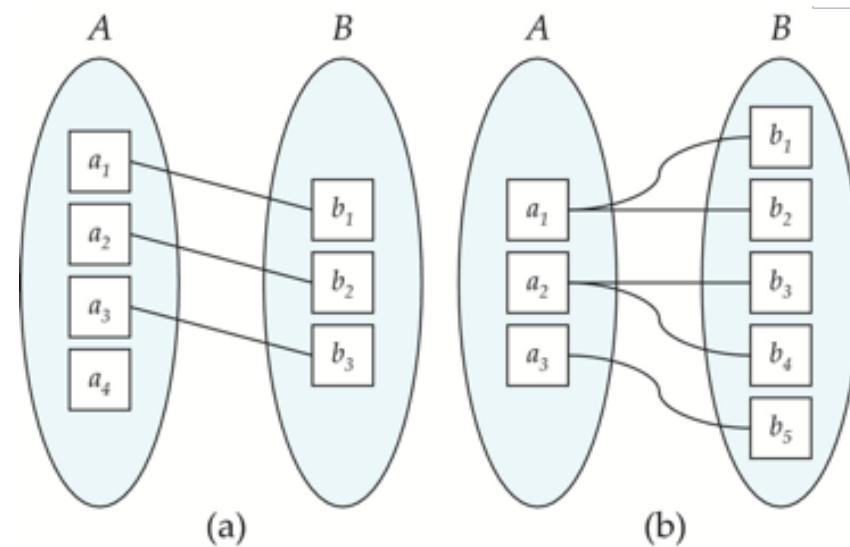
Refresh on ACID Properties

- ACID, which stands for Atomicity, Consistency, Isolation, and Durability[^11]
- **Atomicity** refers to something that cannot be broken down into smaller parts.
 - It is not about concurrency (which comes with the I)
- **Consistency** (overused term), that here relates to the data *invariants* (integrity would be a better term IMHO)
- **Isolation** means that concurrently executing transactions are isolated from each other.
 - Typically associated with serializability, but there weaker options.

Refresh on ACID Properties

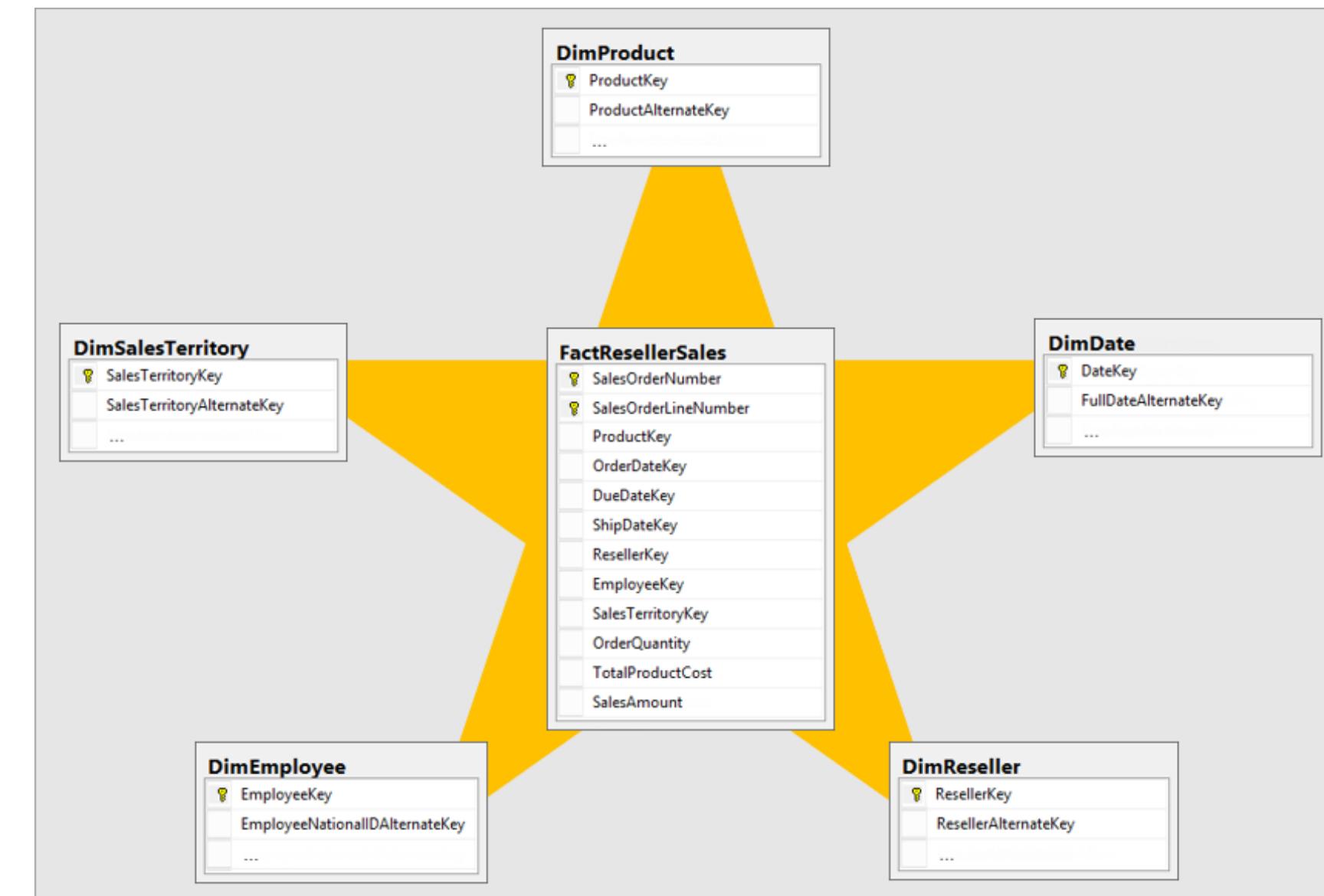
- ACID, which stands for Atomicity, Consistency, Isolation, and Durability[^11]
- **Atomicity** refers to something that cannot be broken down into smaller parts.
 - It is not about concurrency (which comes with the I)
- **Consistency** (overused term), that here relates to the data *invariants* (integrity would be a better term IMHO)
- **Isolation** means that concurrently executing transactions are isolated from each other.
 - Typically associated with serializability, but there weaker options.
- **Durability** means (fault-tolerant) persistency of the data, once the transaction is completed.

Cardinality Visualized



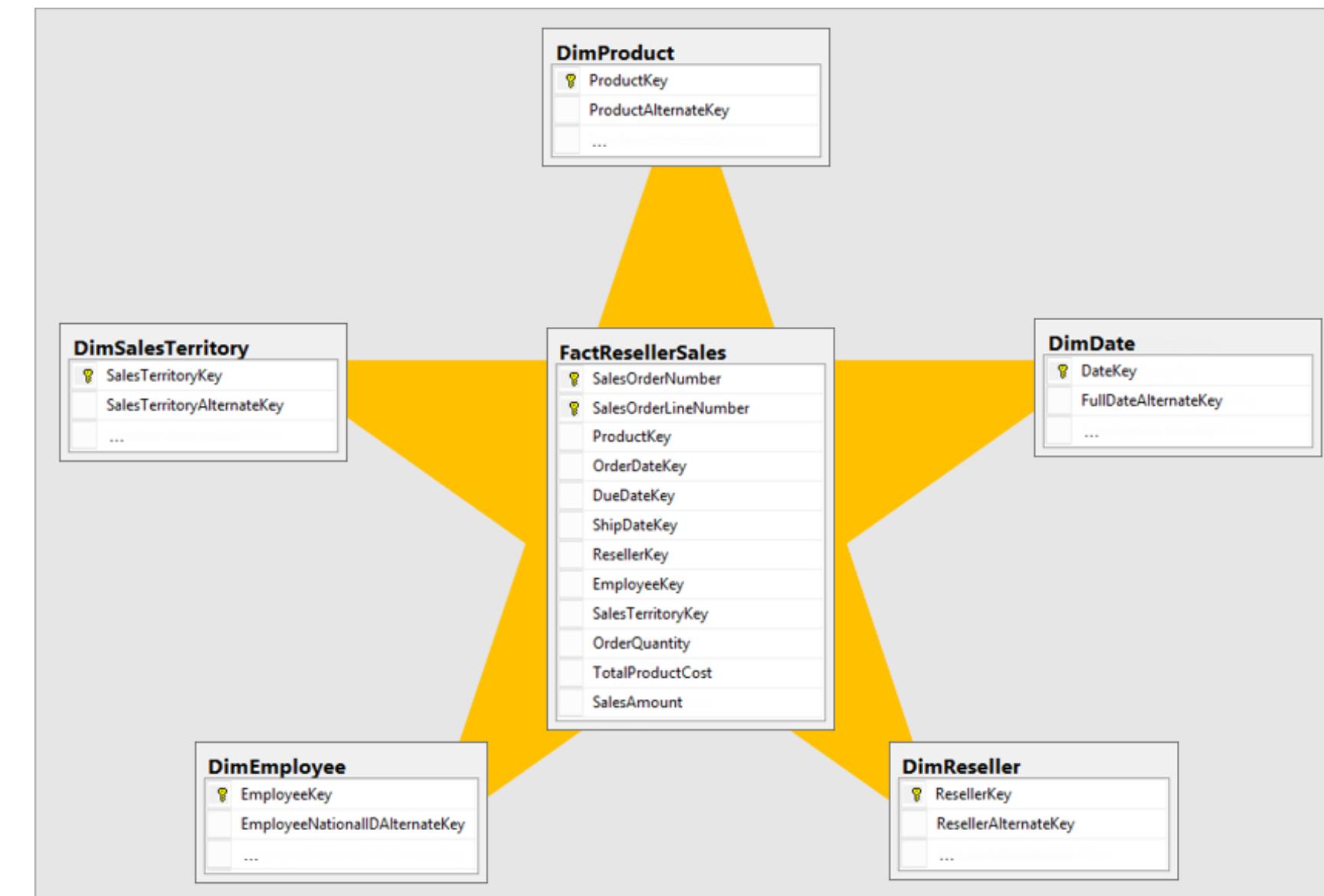
Data Modelling for Data Warehouse (Briefly)

Star Schema



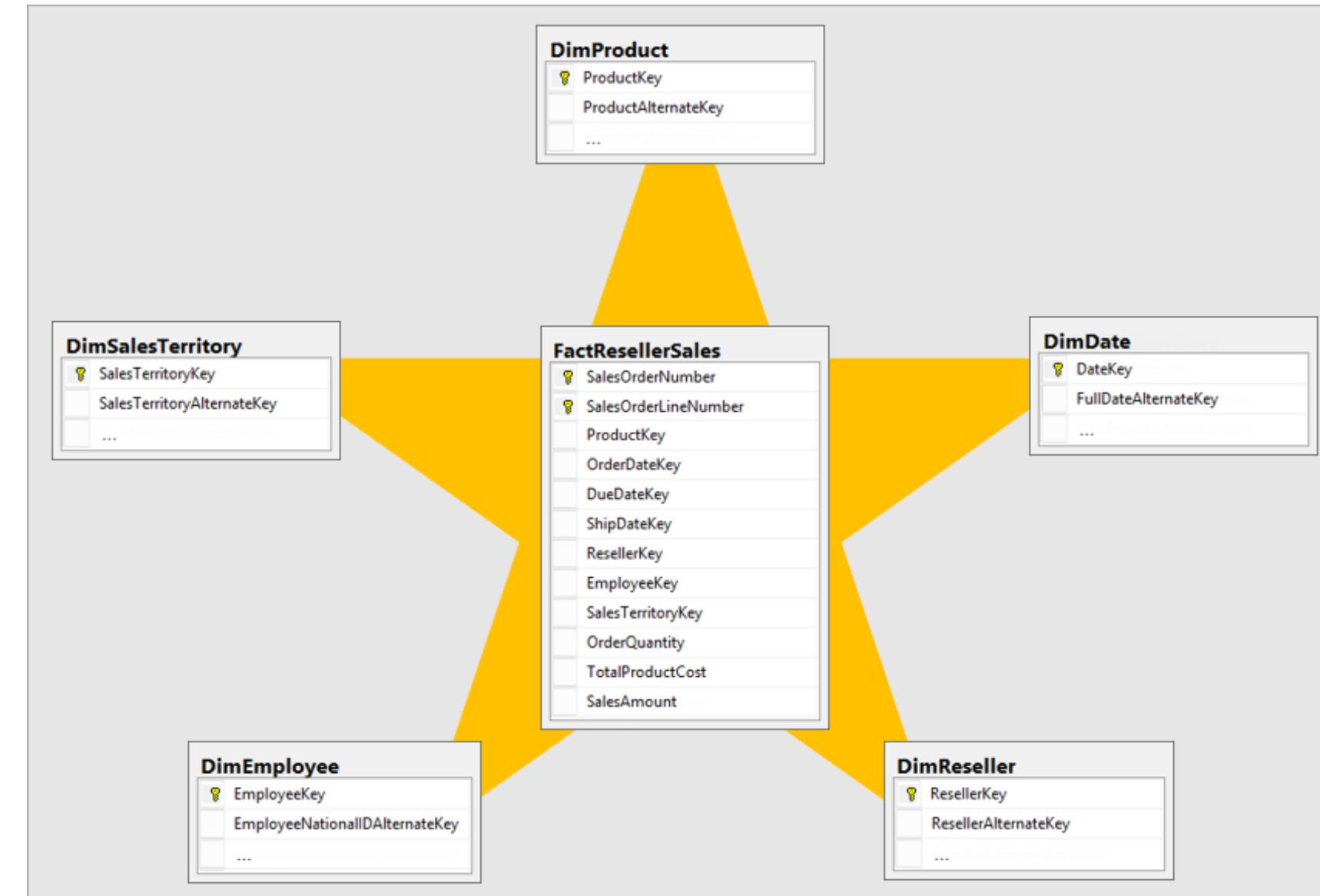
Star Schema

- A **fact table** contains the numeric measures produced by an operational measurement event in the real world.



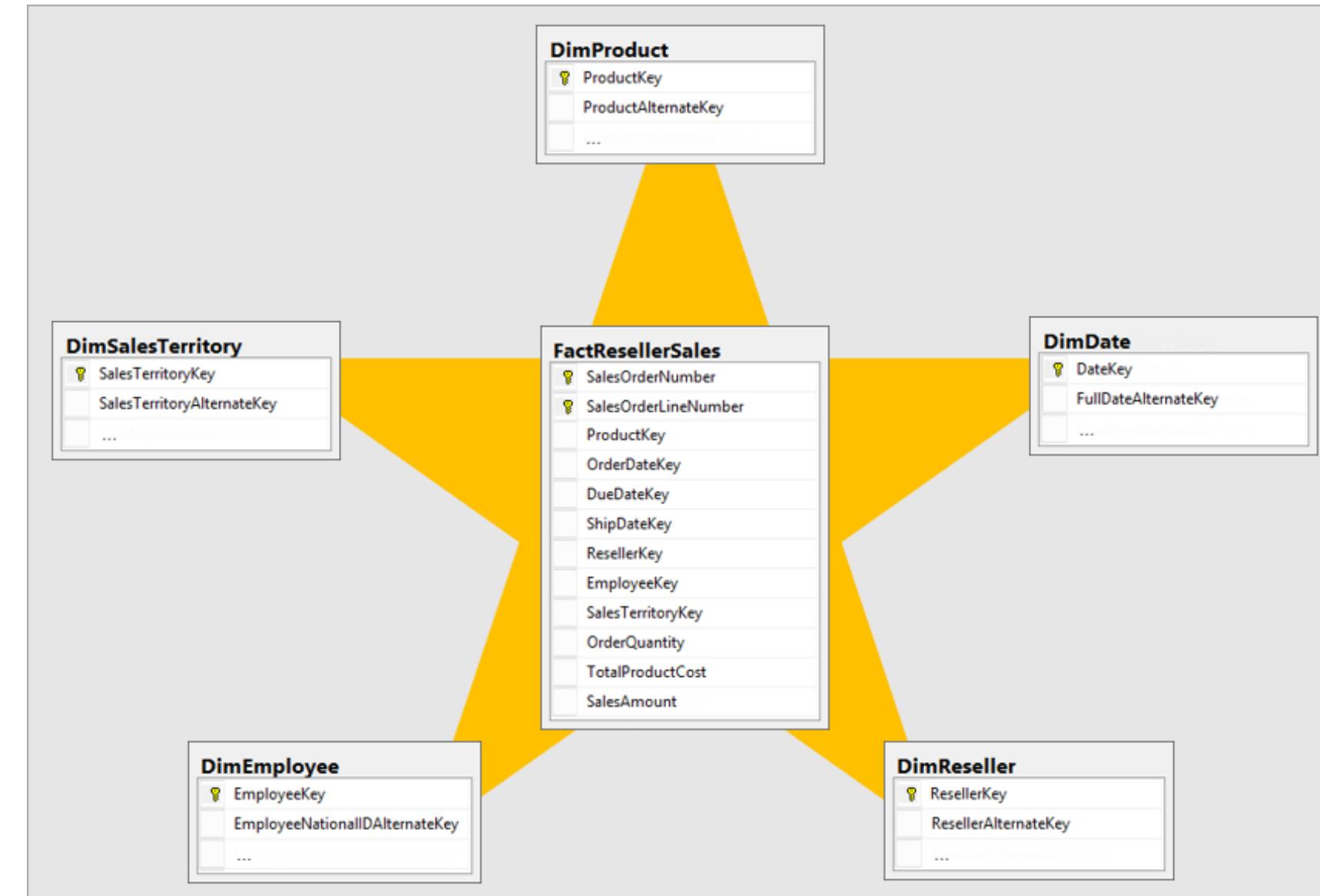
Star Schema

- A **fact table** contains the numeric measures produced by an operational measurement event in the real world.
- A **single fact** table row has a one-to-one relationship to a measurement event as described by the fact table's grain.

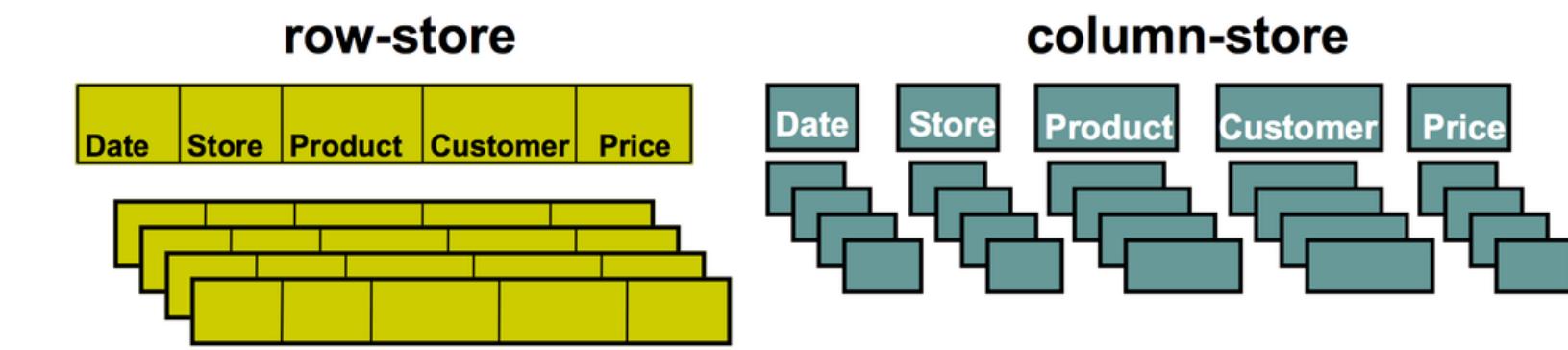


Star Schema

- A **fact table** contains the numeric measures produced by an operational measurement event in the real world.
- A **single fact** table row has a one-to-one relationship to a measurement event as described by the fact table's grain.
- **Dimensions** provide context to business process events, e.g., who, what, where, when, why, and how.

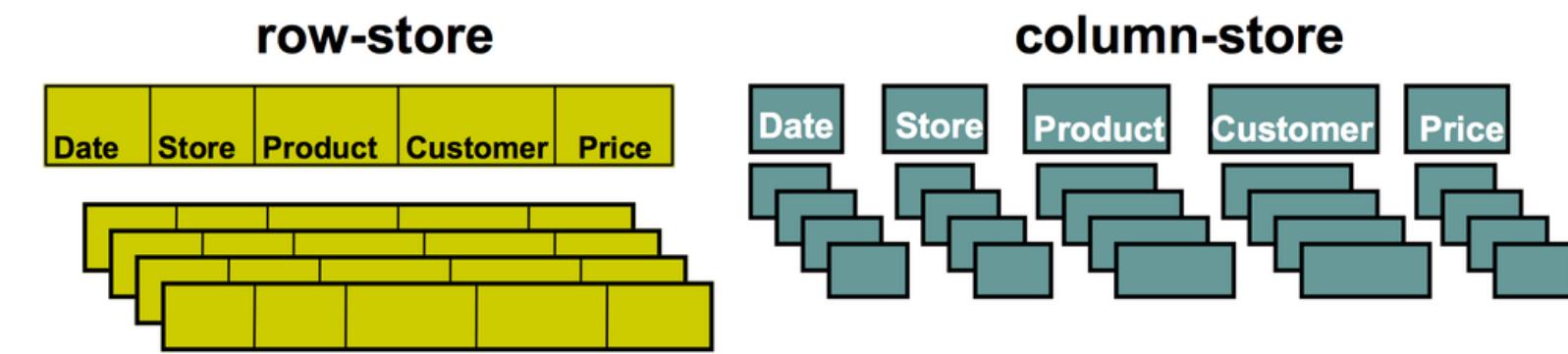


A note on Storage



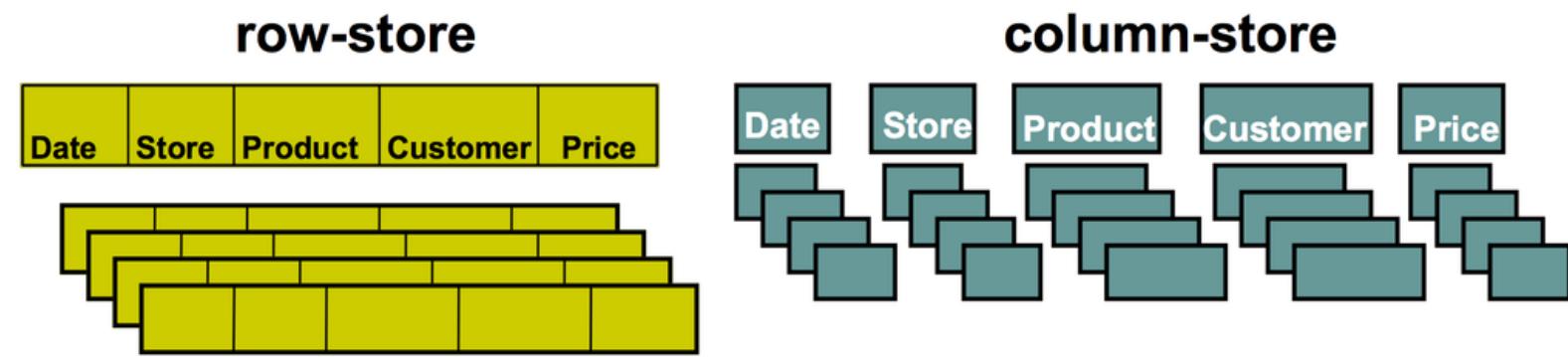
A note on Storage

- Data warehouse typically interact with OLTP database to expose one or more OLAP system.



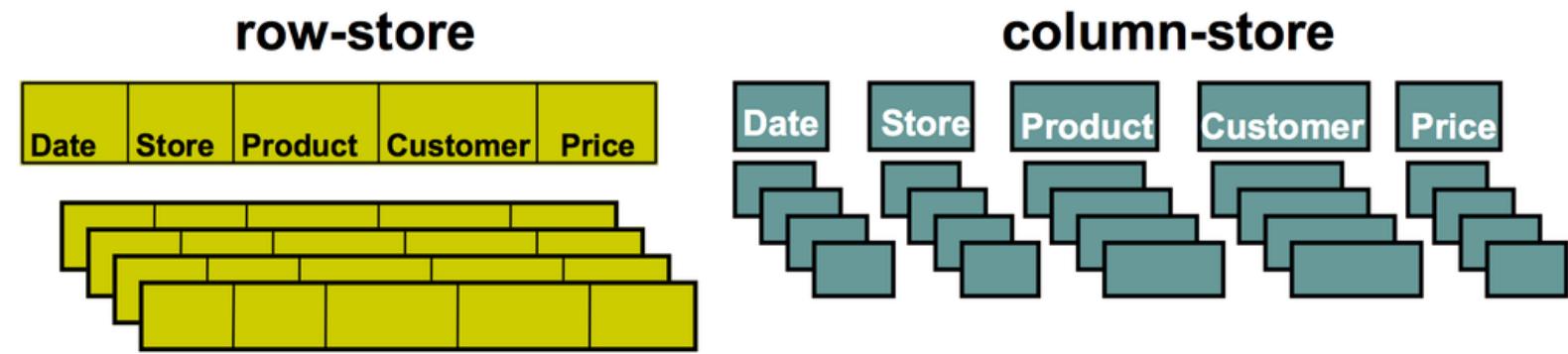
A note on Storage

- Data warehouse typically interact with OLTP database to expose one or more OLAP system.
- Such OLAP system adopt storage optimized for analytics, i.e., Column Oriented



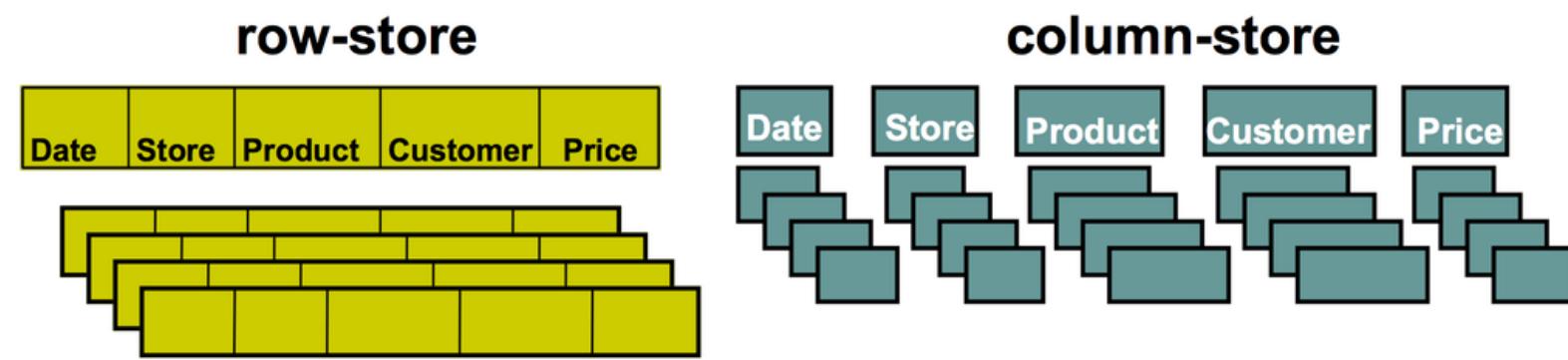
A note on Storage

- Data warehouse typically interact with OLTP database to expose one or more OLAP system.
- Such OLAP system adopt storage optimized for analytics, i.e., Column Oriented
- The column-oriented storage layout relies on each column file containing the rows in the same order.



A note on Storage

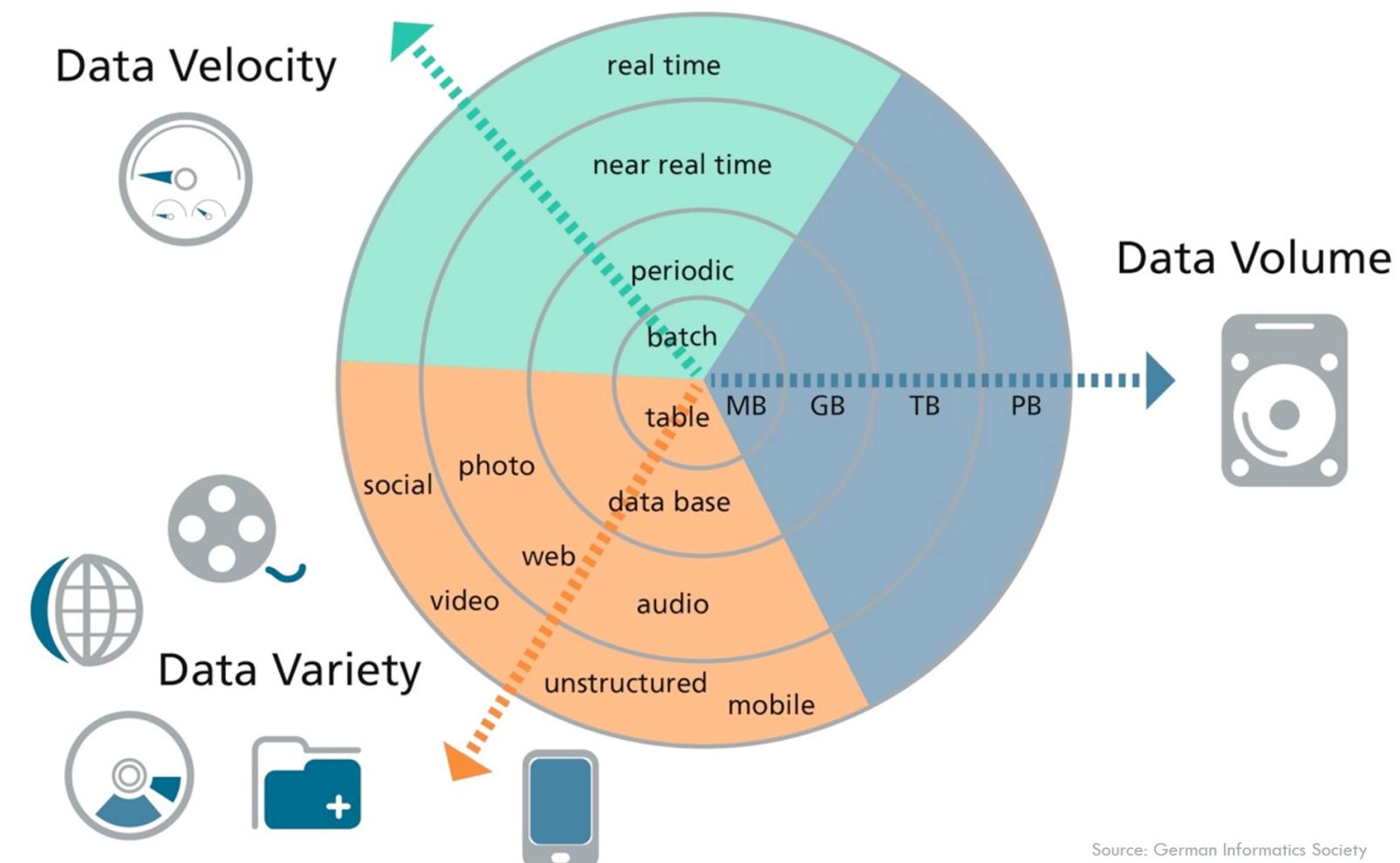
- Data warehouse typically interact with OLTP database to expose one or more OLAP system.
- Such OLAP system adopt storage optimized for analytics, i.e., Column Oriented
- The column-oriented storage layout relies on each column file containing the rows in the same order.
- Not just relational data, e.g., Apache Parquet



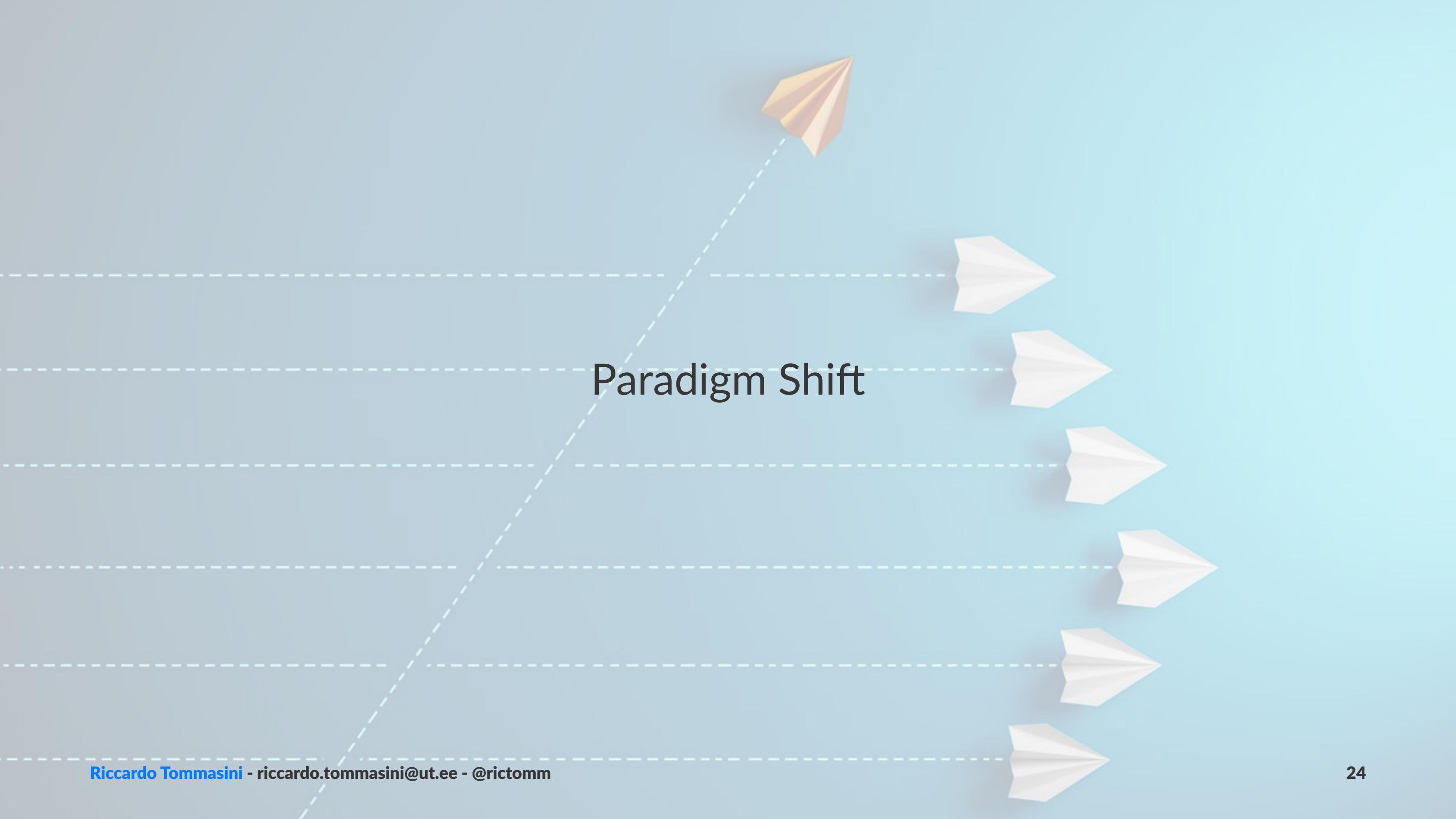
Data Modelling for Big Data

Big Data

Challenges 014



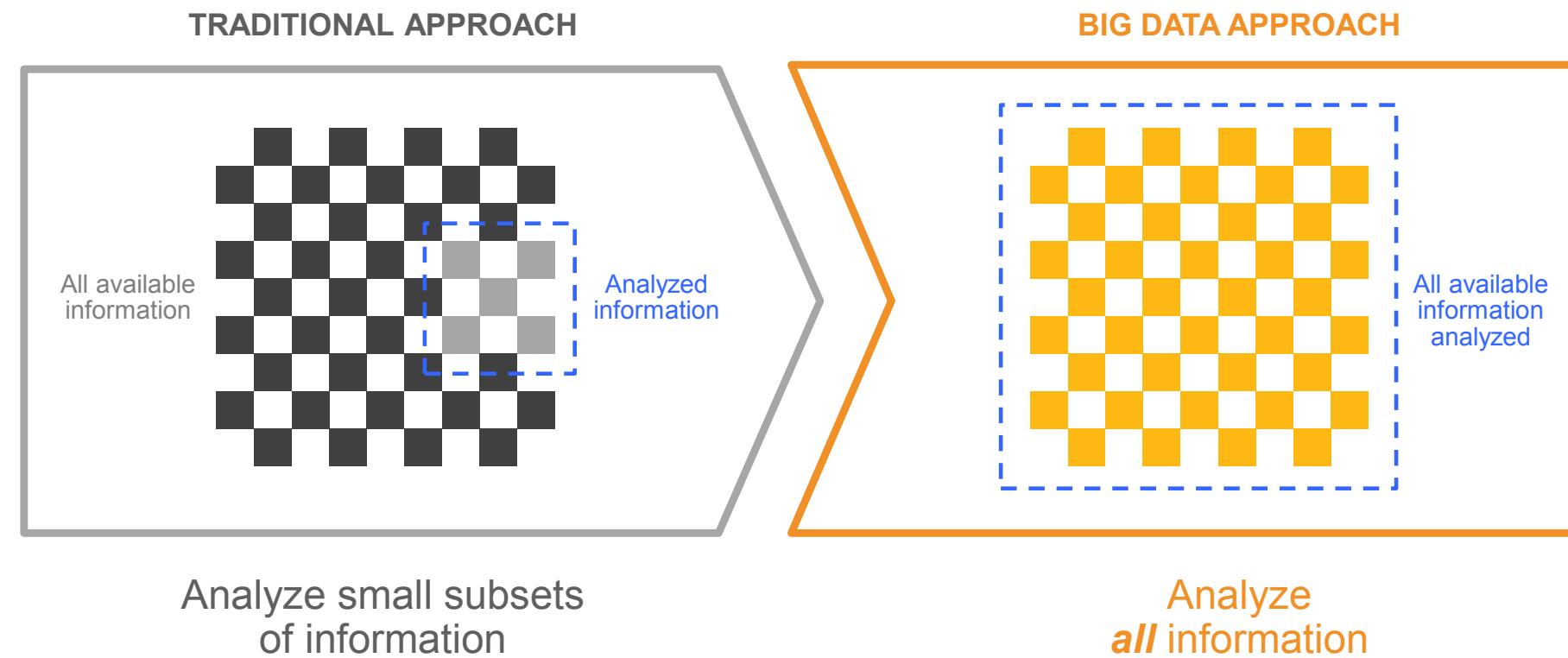
014 Lanely, 2001



Paradigm Shift

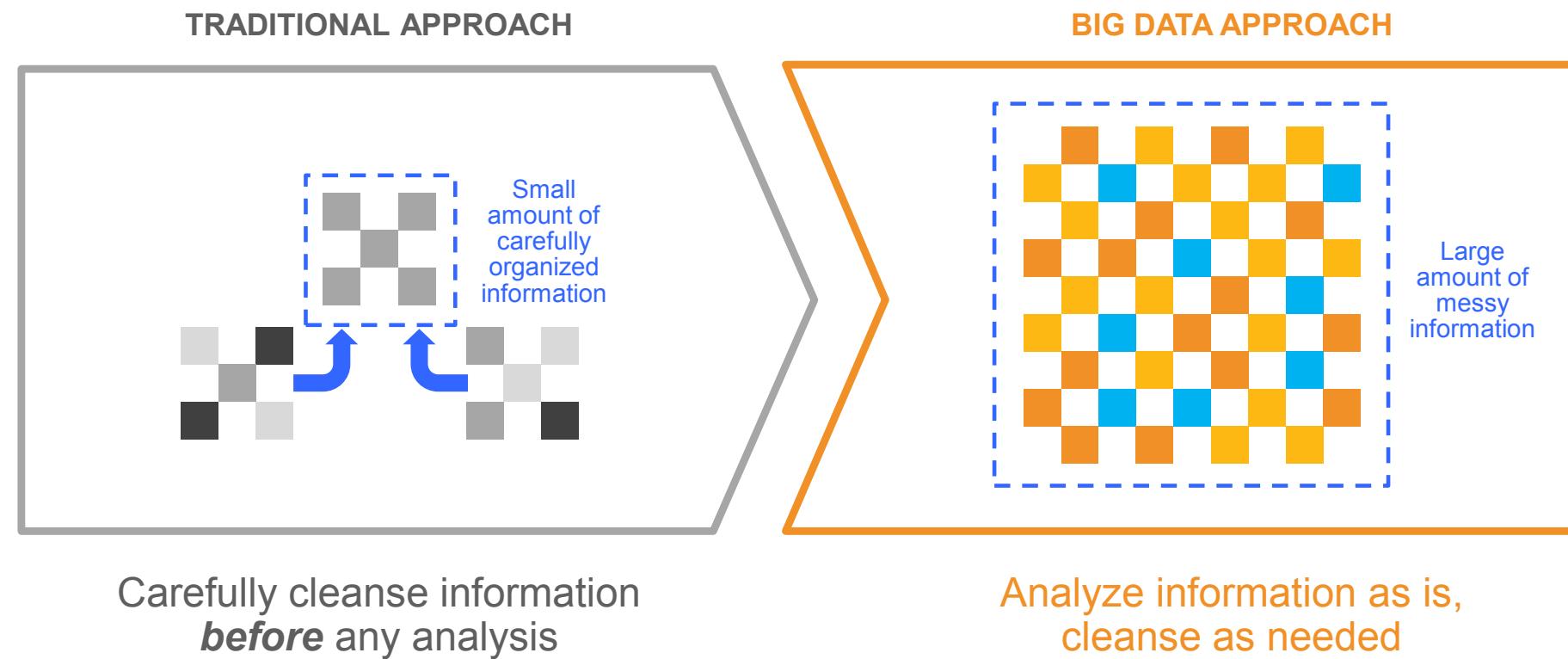
Paradigm shifts enabled by big data

Leverage more of the data being captured



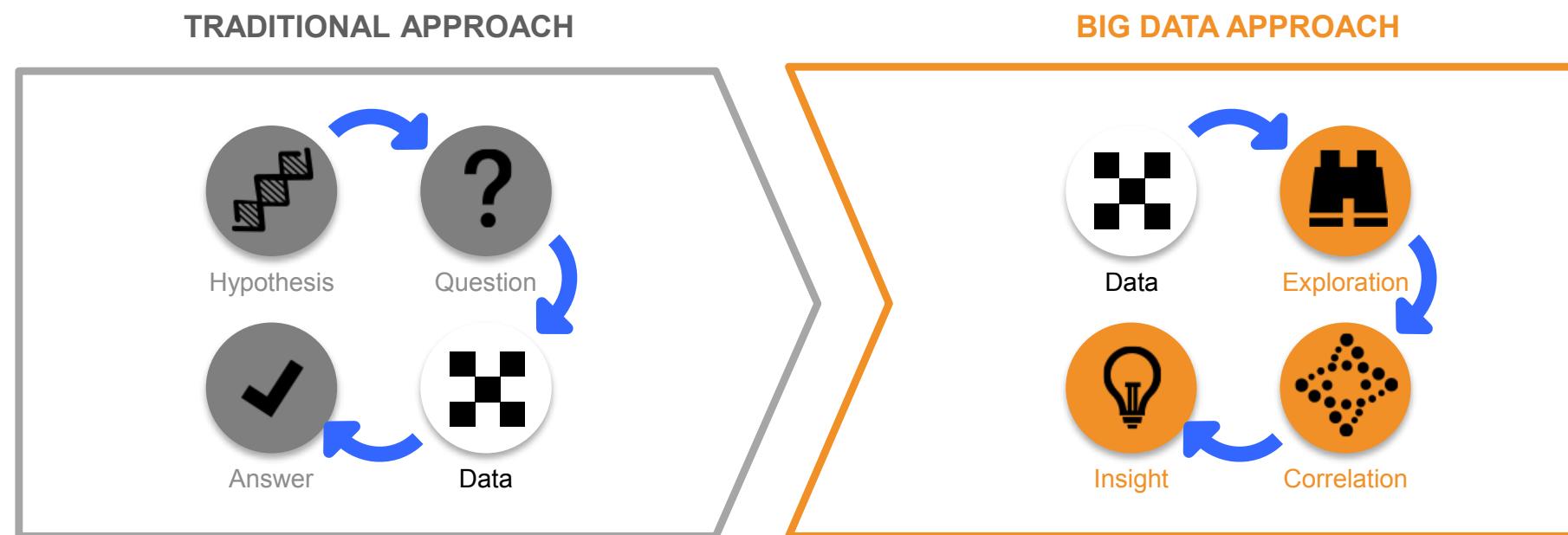
Paradigm shifts enabled by big data

Reduce effort required to leverage data



Paradigm shifts enabled by big data

Data leads the way—and sometimes correlations are good enough



Start with hypothesis and test against selected data

Explore **all** data and identify correlations

Paradigm shifts enabled by big data

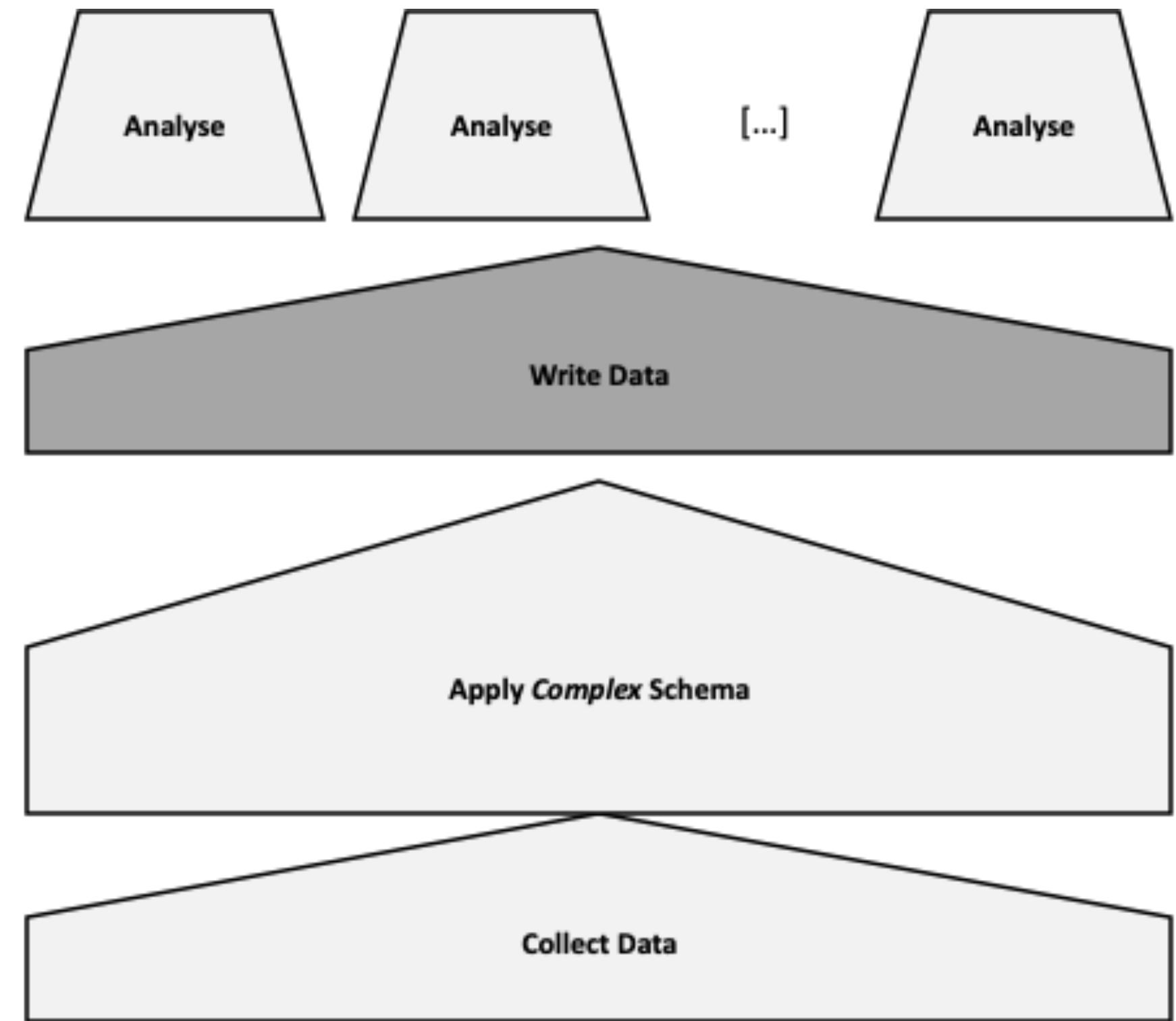
Leverage data as it is captured



Analyze data *after* it's been processed and landed in a warehouse or mart

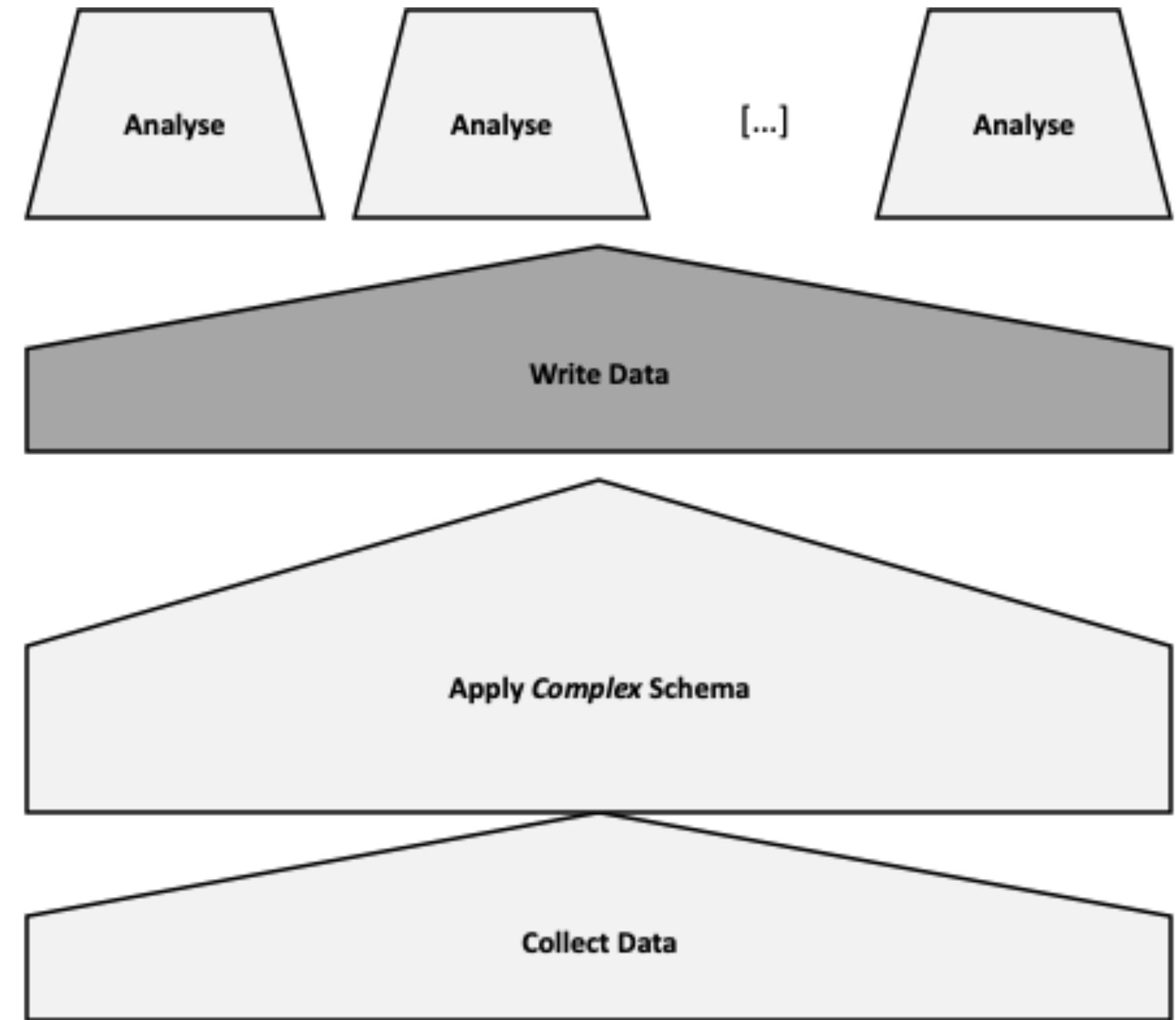
Analyze data *in motion* as it's generated, in real-time

From Schema on Write



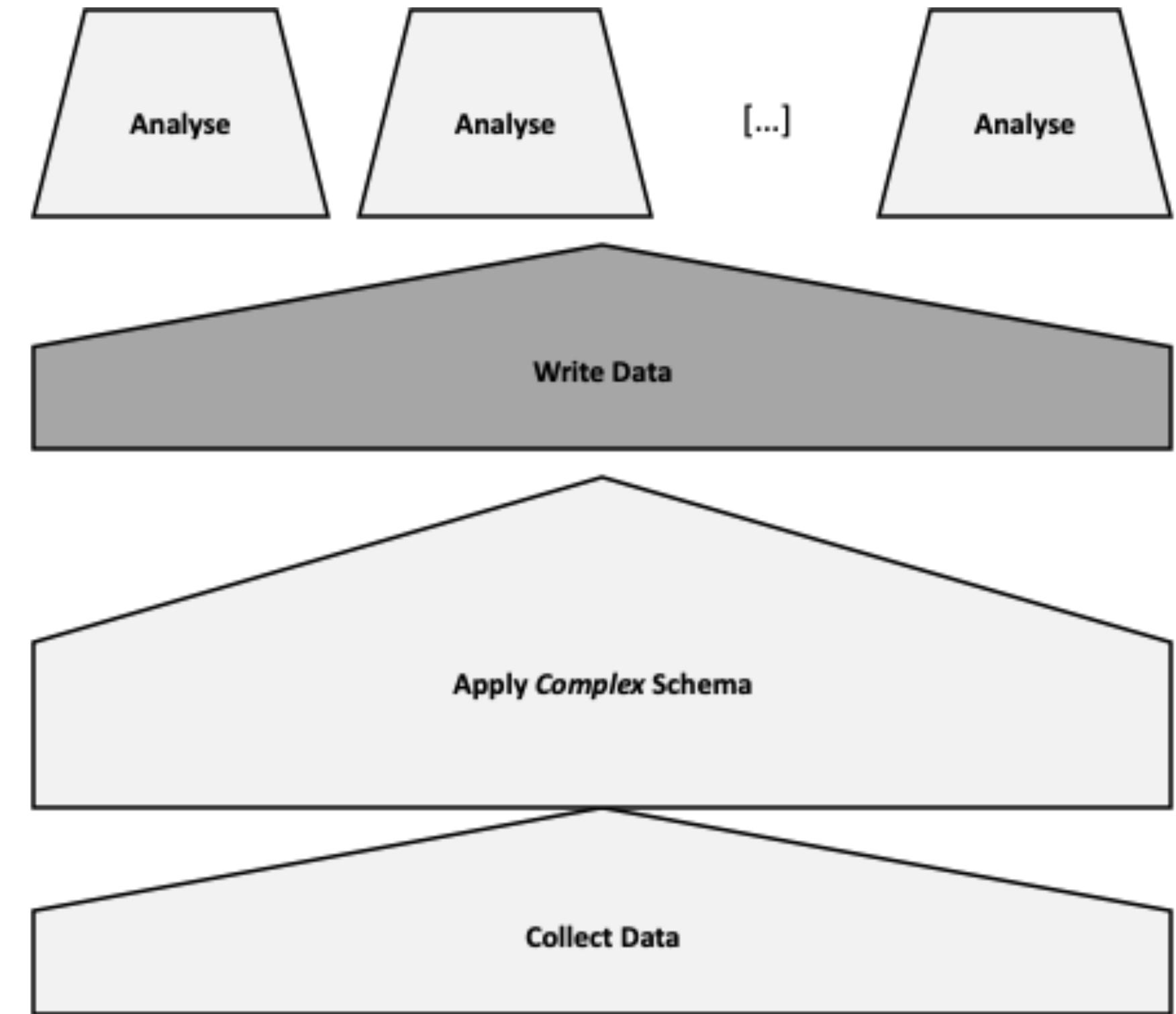
From Schema on Write

- Focus on the modelling a schema that can accommodate all needs

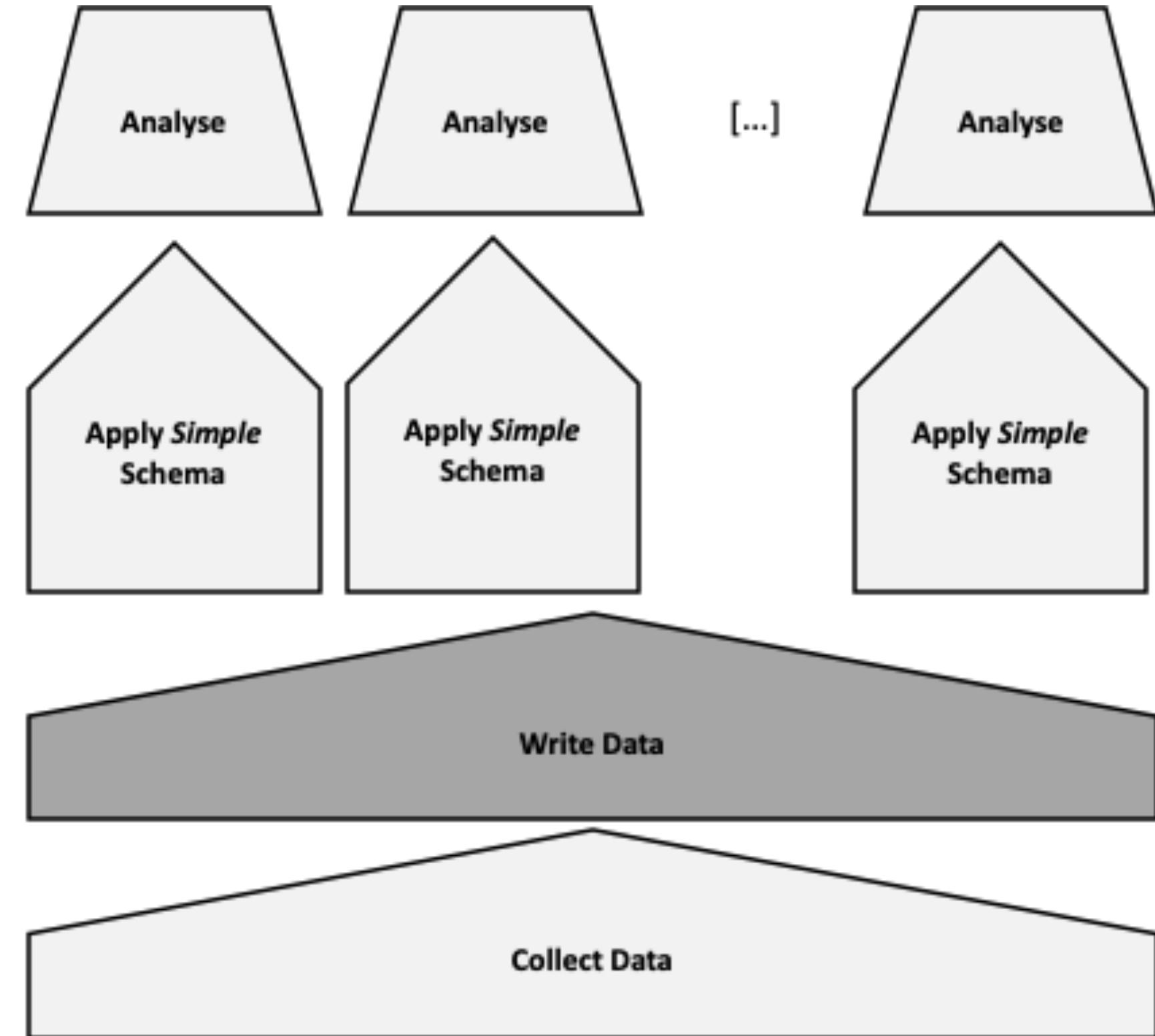


From Schema on Write

- Focus on the modelling a schema that can accommodate all needs
- Bad impact on those analysis that were not envisioned

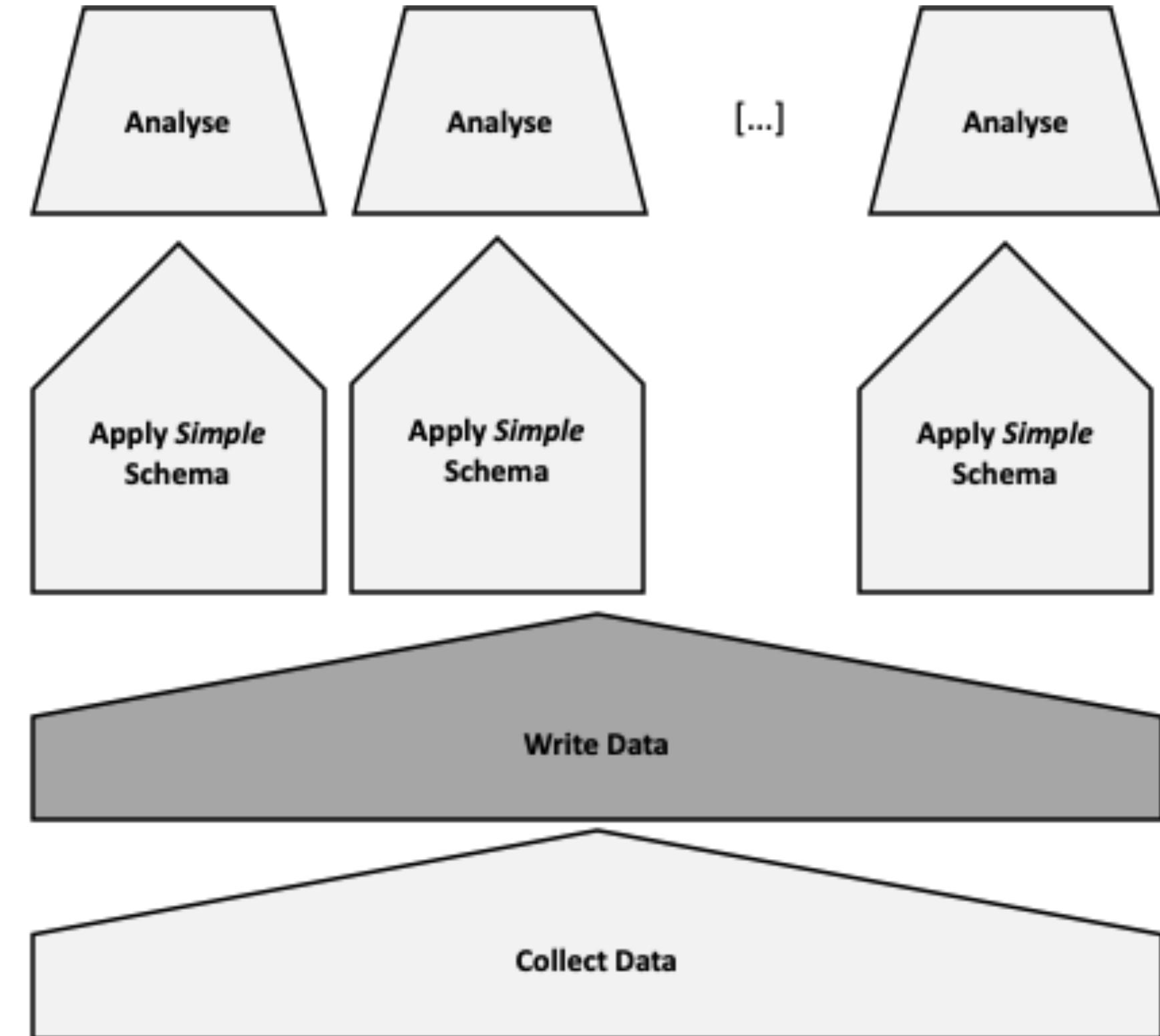


To Schema on Read



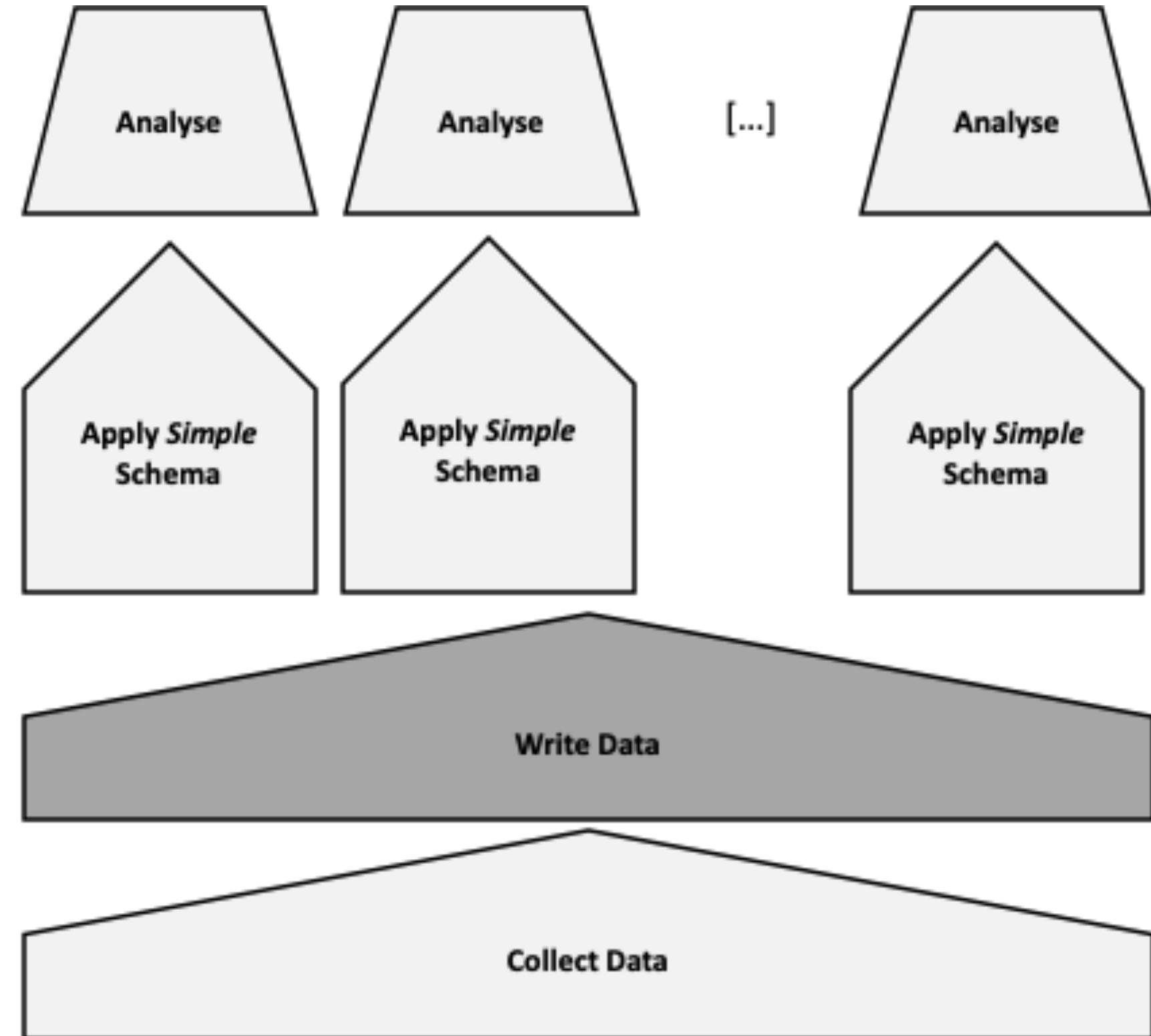
To Schema on Read

- Load data first, ask question later



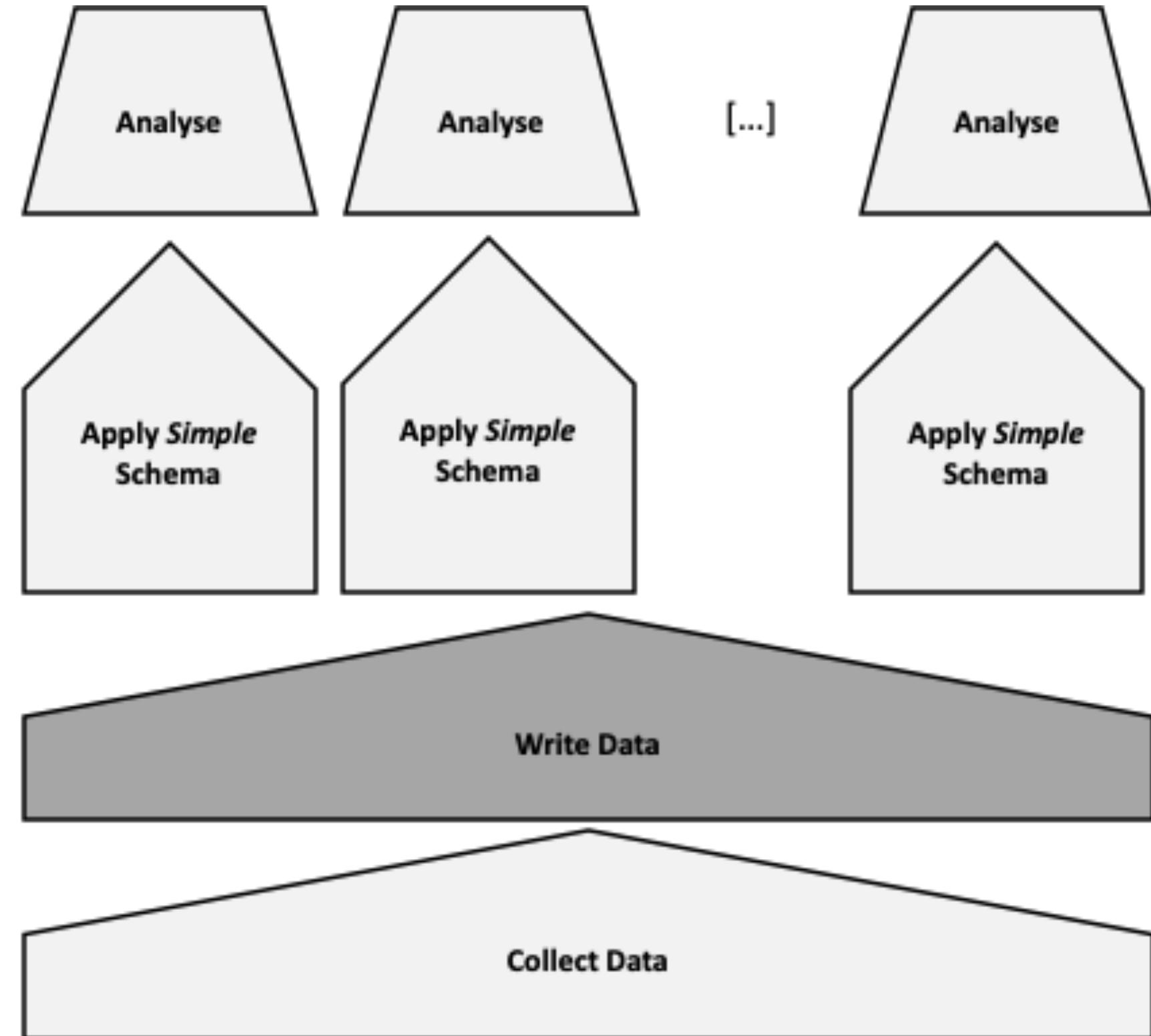
To Schema on Read

- Load data first, ask question later
- All data are kept, the minimal schema need for an analysis is applied when needed



To Schema on Read

- Load data first, ask question later
- All data are kept, the minimal schema need for an analysis is applied when needed
- New analyses can be introduced in any point in time



NoSQL

NoSQL

- **Queryability:** need for specialized query operations that are not well supported by the relational model

NoSQL

- **Queryability:** need for specialized query operations that are not well supported by the relational model
- **Schemaless:** desire for a more dynamic and expressive data model than relational

NoSQL

- **Queryability:** need for specialized query operations that are not well supported by the relational model
- **Schemaless:** desire for a more dynamic and expressive data model than relational
- **Object-Relational Mismatch:** translation between the objects in the application code and the database model

| Document Database | Graph Databases |
|---|---|
|   |   |
| Wide Column Stores | Key-Value Databases |
|   |          |

Kinds of NoSQL (2/4)

NoSQL solutions fall into four major areas:

Kinds of NoSQL (2/4)

NoSQL solutions fall into four major areas:

- **Key-Value Store**

Kinds of NoSQL (2/4)

NoSQL solutions fall into four major areas:

- **Key-Value Store**
 - A key that refers to a payload (actual content / data)

Kinds of NoSQL (2/4)

NoSQL solutions fall into four major areas:

- **Key-Value Store**
 - A key that refers to a payload (actual content / data)
 - Examples: MemcacheDB, Azure Table Storage, Redis, HDFS

Kinds of NoSQL (2/4)

NoSQL solutions fall into four major areas:

- **Key-Value Store**
 - A key that refers to a payload (actual content / data)
 - Examples: MemcacheDB, Azure Table Storage, Redis, HDFS
- **Column Store**

Kinds of NoSQL (2/4)

NoSQL solutions fall into four major areas:

- **Key-Value Store**

- A key that refers to a payload (actual content / data)
- Examples: MemcacheDB, Azure Table Storage, Redis, HDFS

- **Column Store**

- Column data is saved together, as opposed to row data

Kinds of NoSQL (2/4)

NoSQL solutions fall into four major areas:

- **Key-Value Store**

- A key that refers to a payload (actual content / data)
- Examples: MemcacheDB, Azure Table Storage, Redis, HDFS

- **Column Store**

- Column data is saved together, as opposed to row data
- Super useful for data analytics

Kinds of NoSQL (2/4)

NoSQL solutions fall into four major areas:

- **Key-Value Store**

- A key that refers to a payload (actual content / data)
- Examples: MemcacheDB, Azure Table Storage, Redis, HDFS

- **Column Store**

- Column data is saved together, as opposed to row data
- Super useful for data analytics
- Examples: Hadoop, Cassandra, Hypertable

Kinds of NoSQL (4/4)

Kinds of NoSQL (4/4)

- **Document / XML / Object Store**

Kinds of NoSQL (4/4)

- **Document / XML / Object Store**
 - Key (and possibly other indexes) point at a serialized object

Kinds of NoSQL (4/4)

- **Document / XML / Object Store**
 - Key (and possibly other indexes) point at a serialized object
 - DB can operate against values in document

Kinds of NoSQL (4/4)

- **Document / XML / Object Store**
 - Key (and possibly other indexes) point at a serialized object
 - DB can operate against values in document
 - Examples: MongoDB, CouchDB, RavenDB

Kinds of NoSQL (4/4)

- **Document / XML / Object Store**
 - Key (and possibly other indexes) point at a serialized object
 - DB can operate against values in document
 - Examples: MongoDB, CouchDB, RavenDB
- **Graph Store**

Kinds of NoSQL (4/4)

- **Document / XML / Object Store**
 - Key (and possibly other indexes) point at a serialized object
 - DB can operate against values in document
 - Examples: MongoDB, CouchDB, RavenDB
- **Graph Store**
 - Nodes are stored independently, and the relationship between nodes (edges) are stored with data

Kinds of NoSQL (4/4)

- **Document / XML / Object Store**
 - Key (and possibly other indexes) point at a serialized object
 - DB can operate against values in document
 - Examples: MongoDB, CouchDB, RavenDB
- **Graph Store**
 - Nodes are stored independently, and the relationship between nodes (edges) are stored with data
 - Examples: AllegroGraph, Neo4j

ACID vs. BASE trade-off

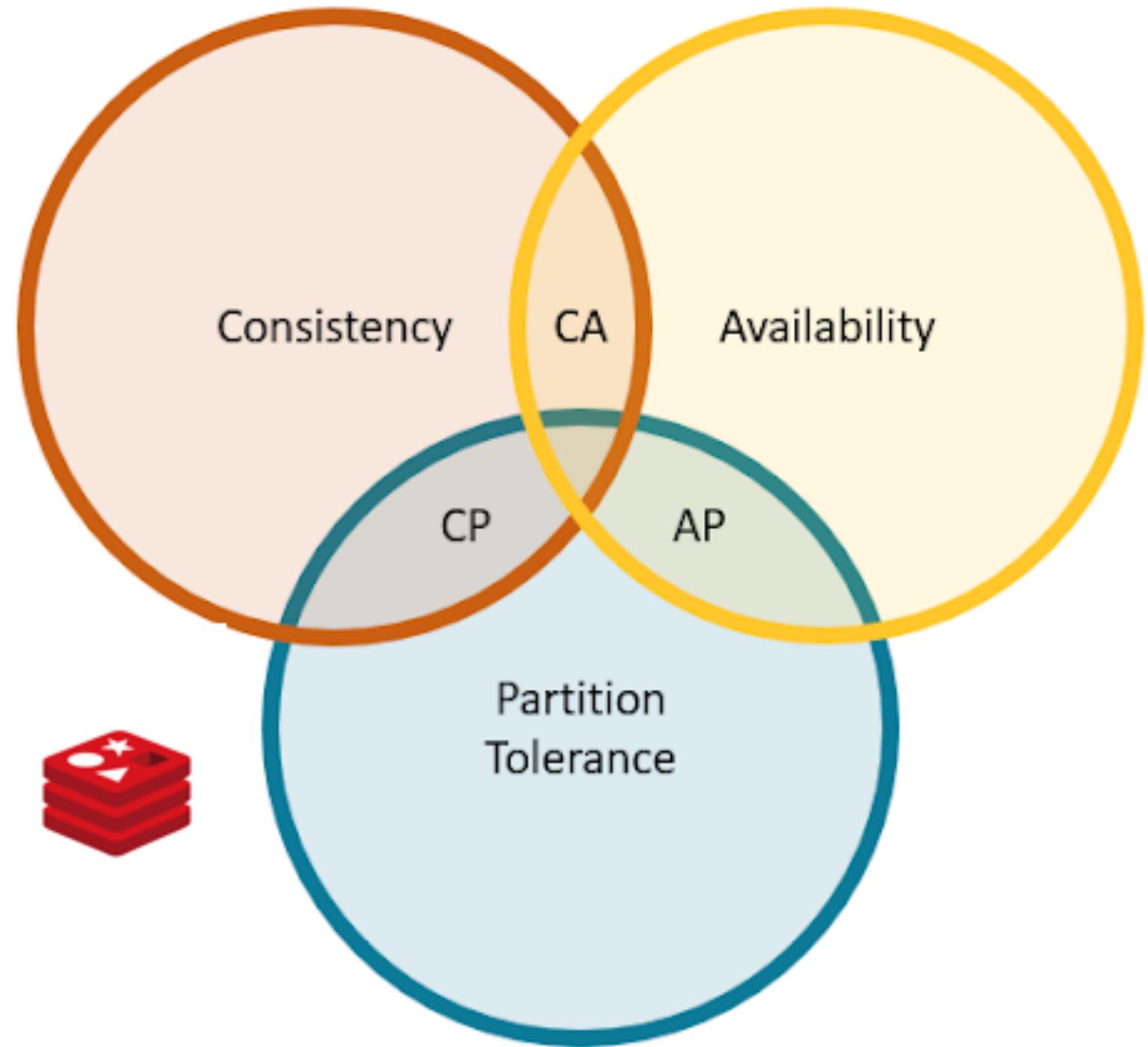
No general answer to whether your application needs an ACID versus BASE consistency model.

Given **BASE**'s loose consistency, developers **need to** be more knowledgeable and **rigorous** about **consistent** data if they choose a BASE store for their application.

Planning around **BASE** limitations can sometimes be a major **disadvantage** when compared to the simplicity of ACID transactions.

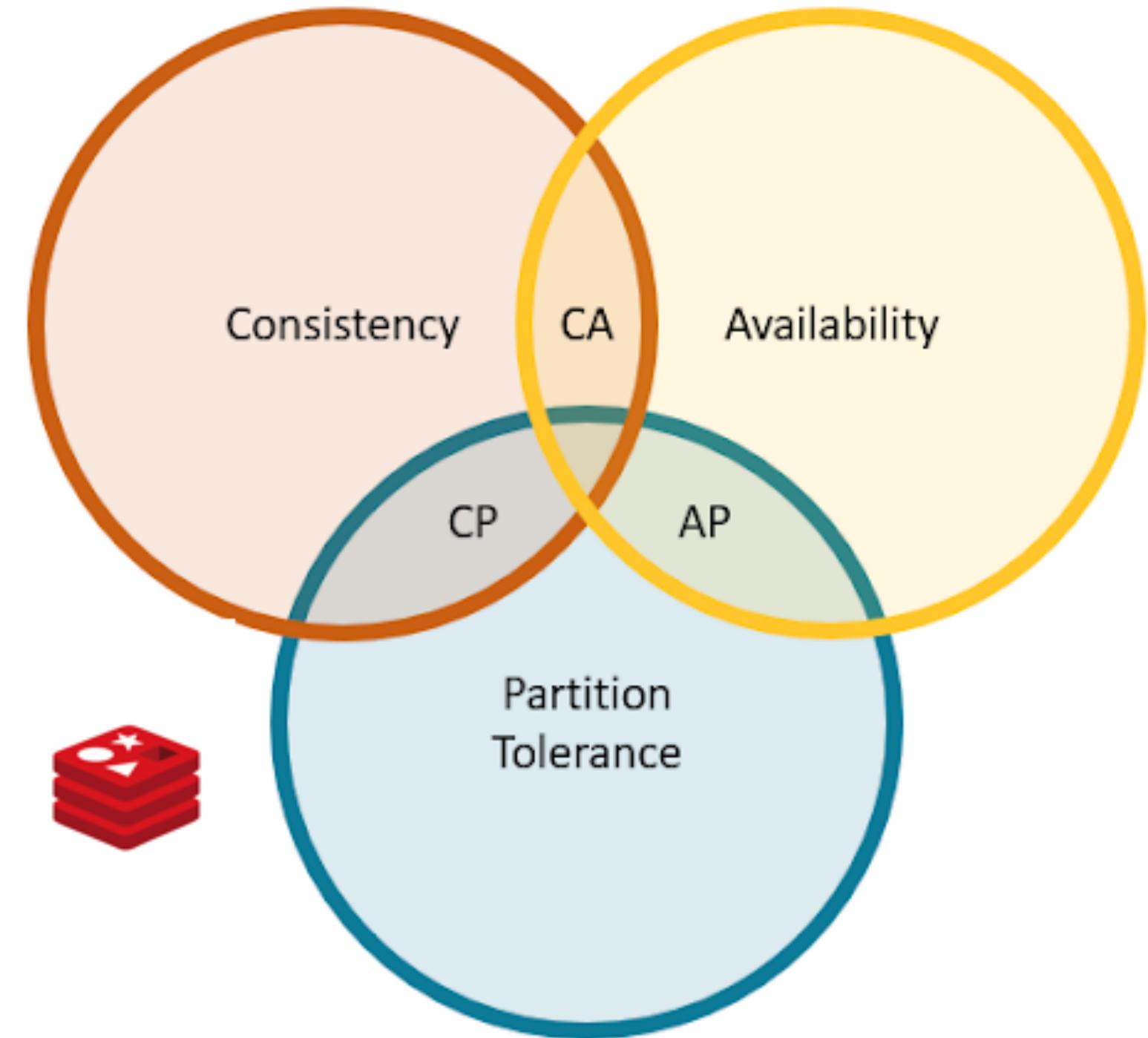
A fully **ACID** database is the perfect fit for use cases where data **reliability** and **consistency** are essential.

Redis



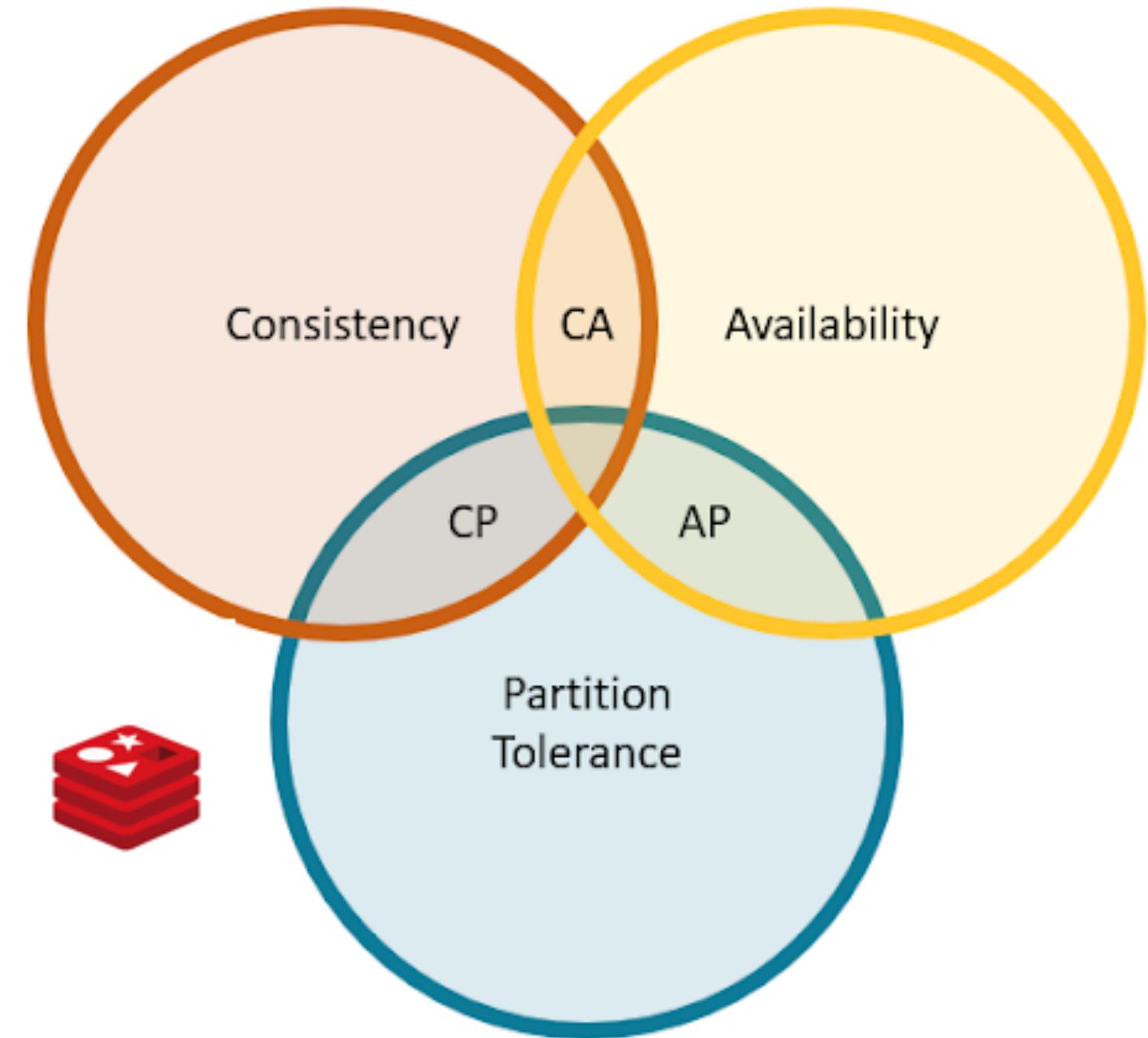
Redis

- is an In-Memory [[Key-Value Store]]



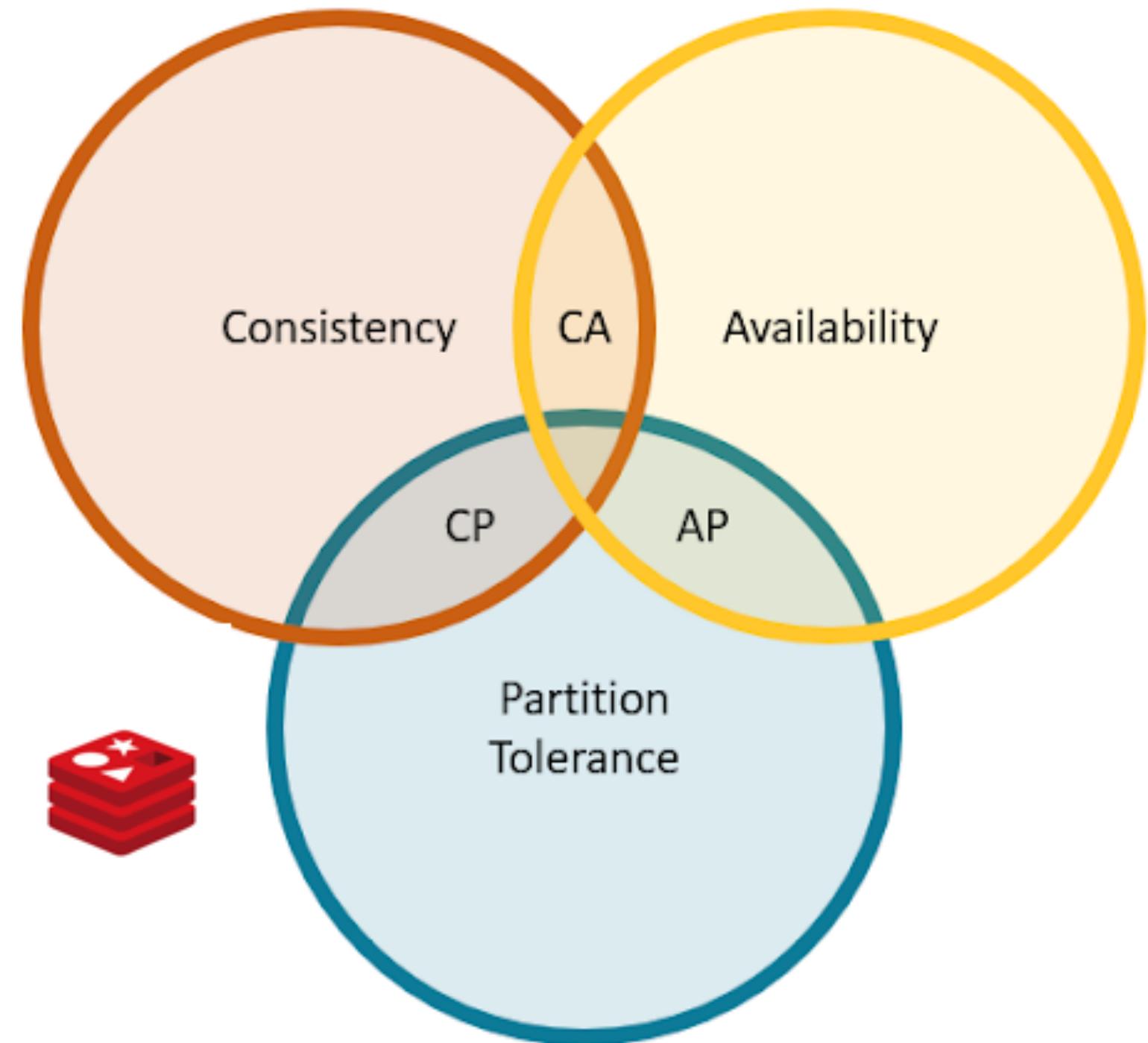
Redis

- is an In-Memory [[Key-Value Store]]
- Strong consistency (**C**)



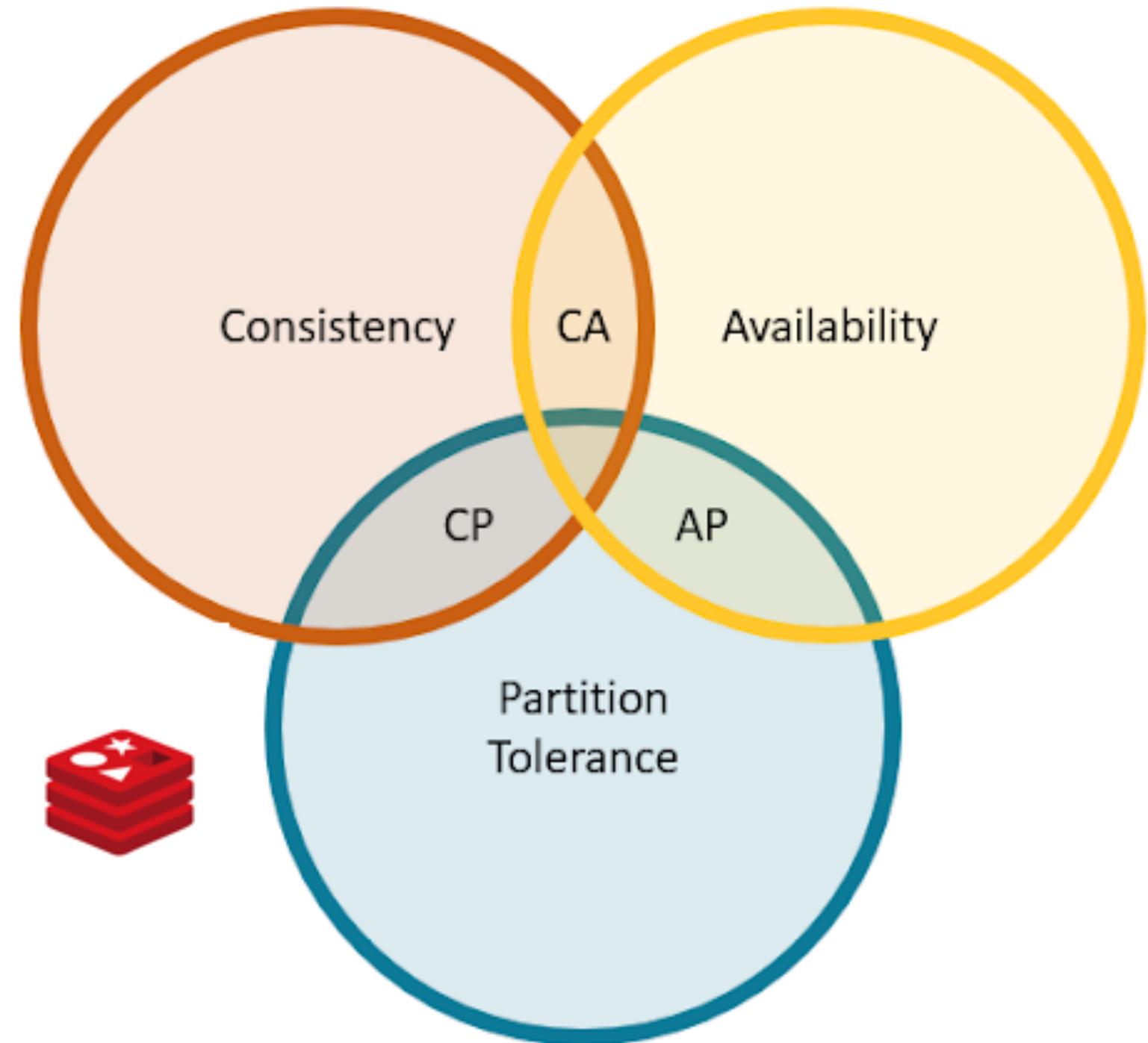
Redis

- is an In-Memory [[Key-Value Store]]
- Strong consistency (**C**)
- *Tuneably available* (**A**)



Redis

- is an In-Memory [[Key-Value Store]]
- Strong consistency (**C**)
- *Tuneably* available (**A**)
- Horizontal Scalable (**P**)



What to remember

What to remember

- Data Model: Hash Table

What to remember

- Data Model: Hash Table
- Data Types: ASCII and Collections

What to remember

- Data Model: Hash Table
- Data Types: ASCII and Collections
- Operations: Lookups and Iterations

What to remember

- Data Model: Hash Table
- Data Types: ASCII and Collections
- Operations: Lookups and Iterations
- Replication: Master-Slaves(read only)

What to remember

- Data Model: Hash Table
- Data Types: ASCII and Collections
- Operations: Lookups and Iterations
- Replication: Master-Slaves(read only)
- Partitioning: Client/Proxy/Query Router

What to remember

- Data Model: Hash Table
- Data Types: ASCII and Collections
- Operations: Lookups and Iterations
- Replication: Master-Slaves(read only)
- Partitioning: Client/Proxy/Query Router
- Architecture

What to remember

- Data Model: Hash Table
- Data Types: ASCII and Collections
- Operations: Lookups and Iterations
- Replication: Master-Slaves(read only)
- Partitioning: Client/Proxy/Query Router
- Architecture
 - Standalone

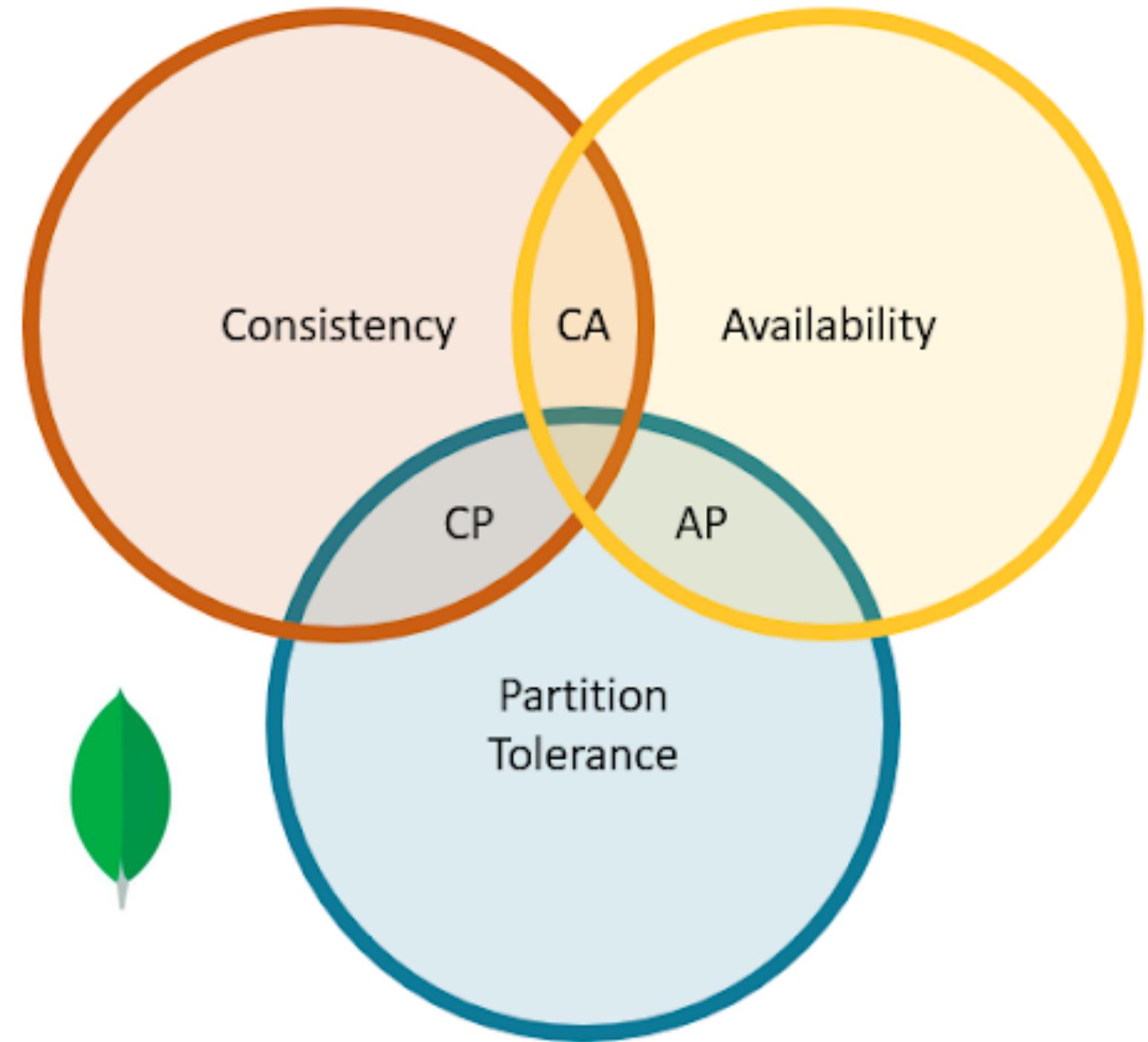
What to remember

- Data Model: Hash Table
- Data Types: ASCII and Collections
- Operations: Lookups and Iterations
- Replication: Master-Slaves(read only)
- Partitioning: Client/Proxy/Query Router
- Architecture
 - Standalone
 - Sentinel

What to remember

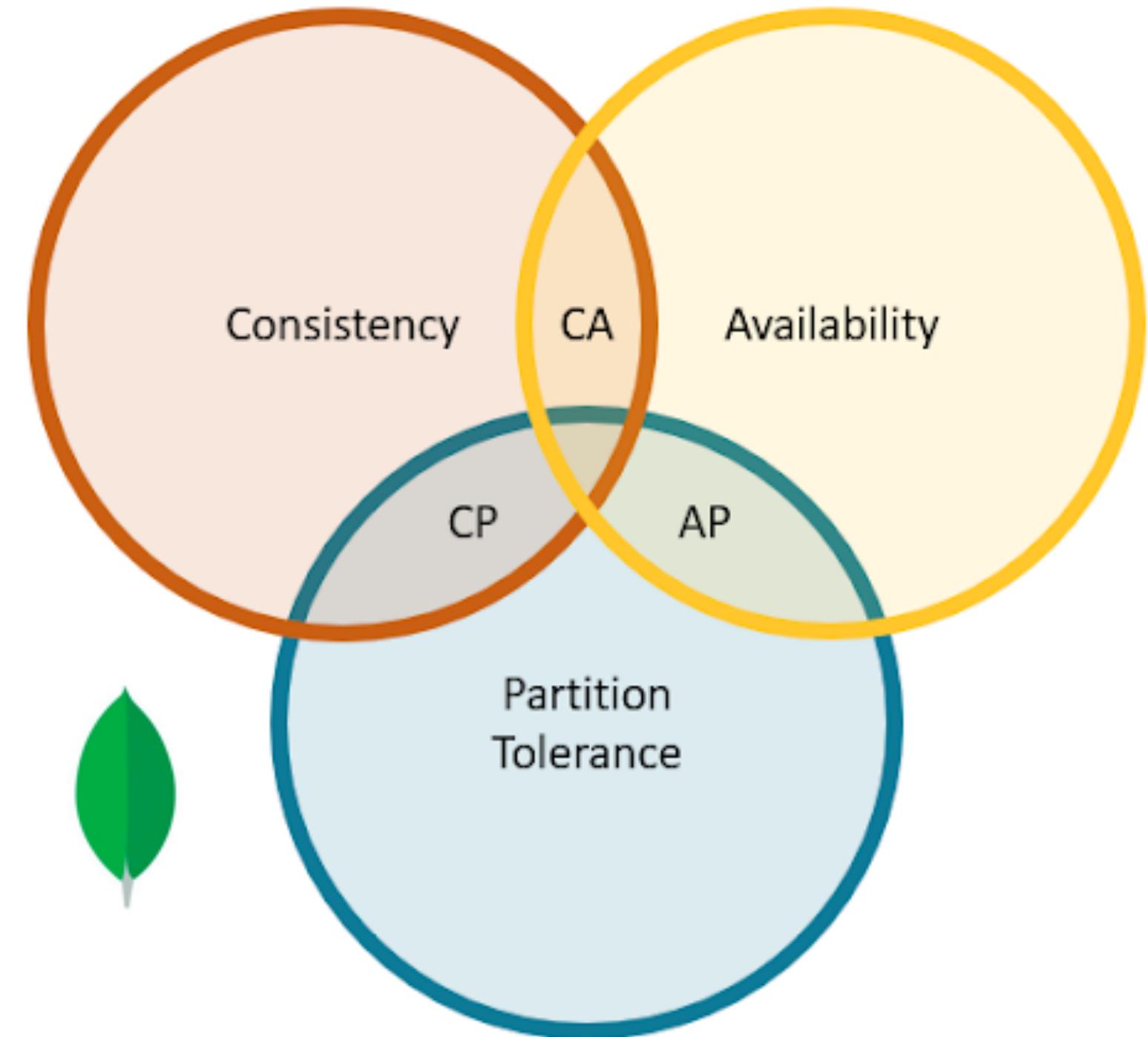
- Data Model: Hash Table
- Data Types: ASCII and Collections
- Operations: Lookups and Iterations
- Replication: Master-Slaves(read only)
- Partitioning: Client/Proxy/Query Router
- Architecture
 - Standalone
 - Sentinel
 - Cluster

MongoDB



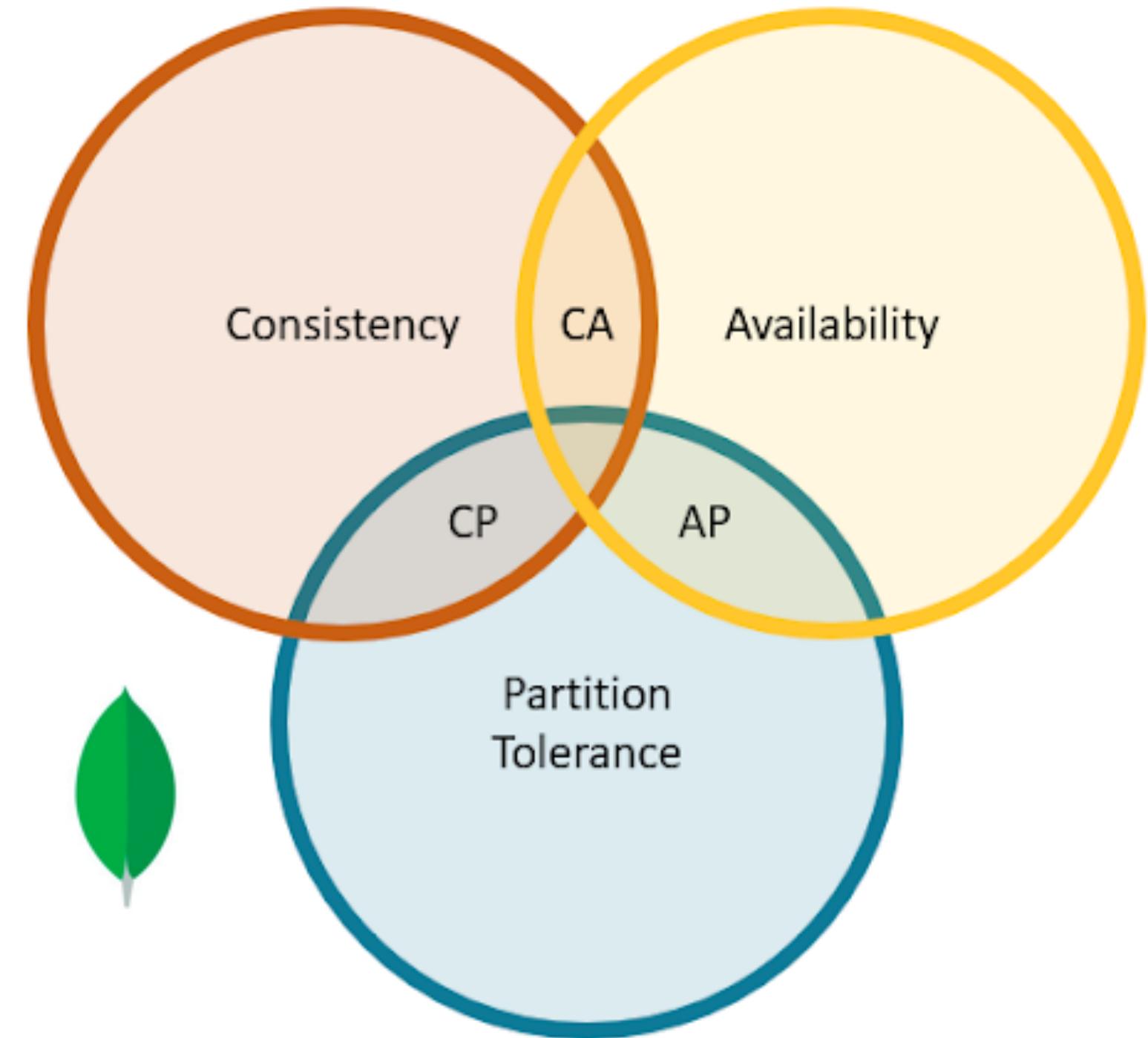
MongoDB

- An In-Memory [[Document Databases]]



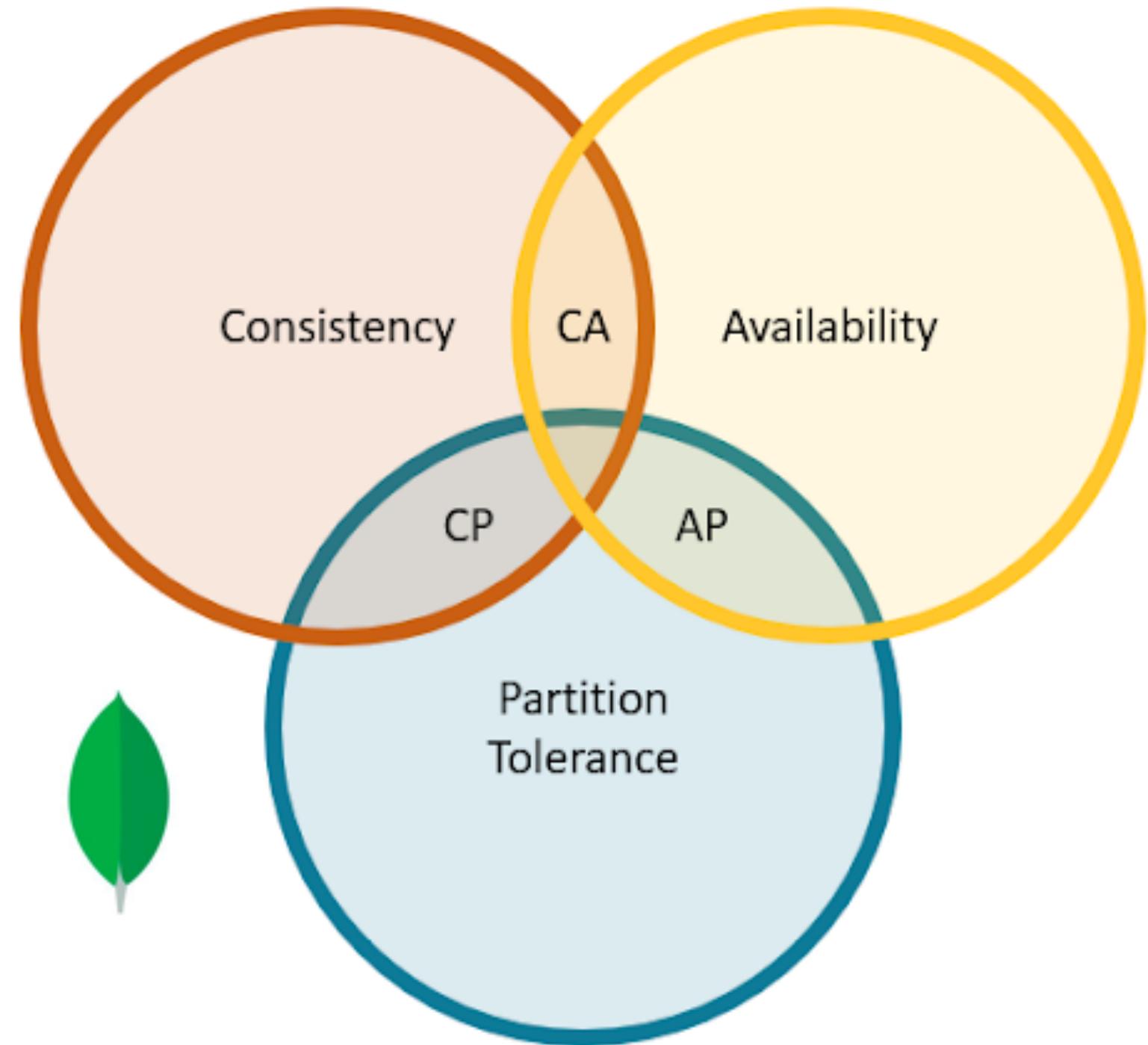
MongoDB

- An In-Memory [[Document Databases]]
- Strong consistency (**C**)



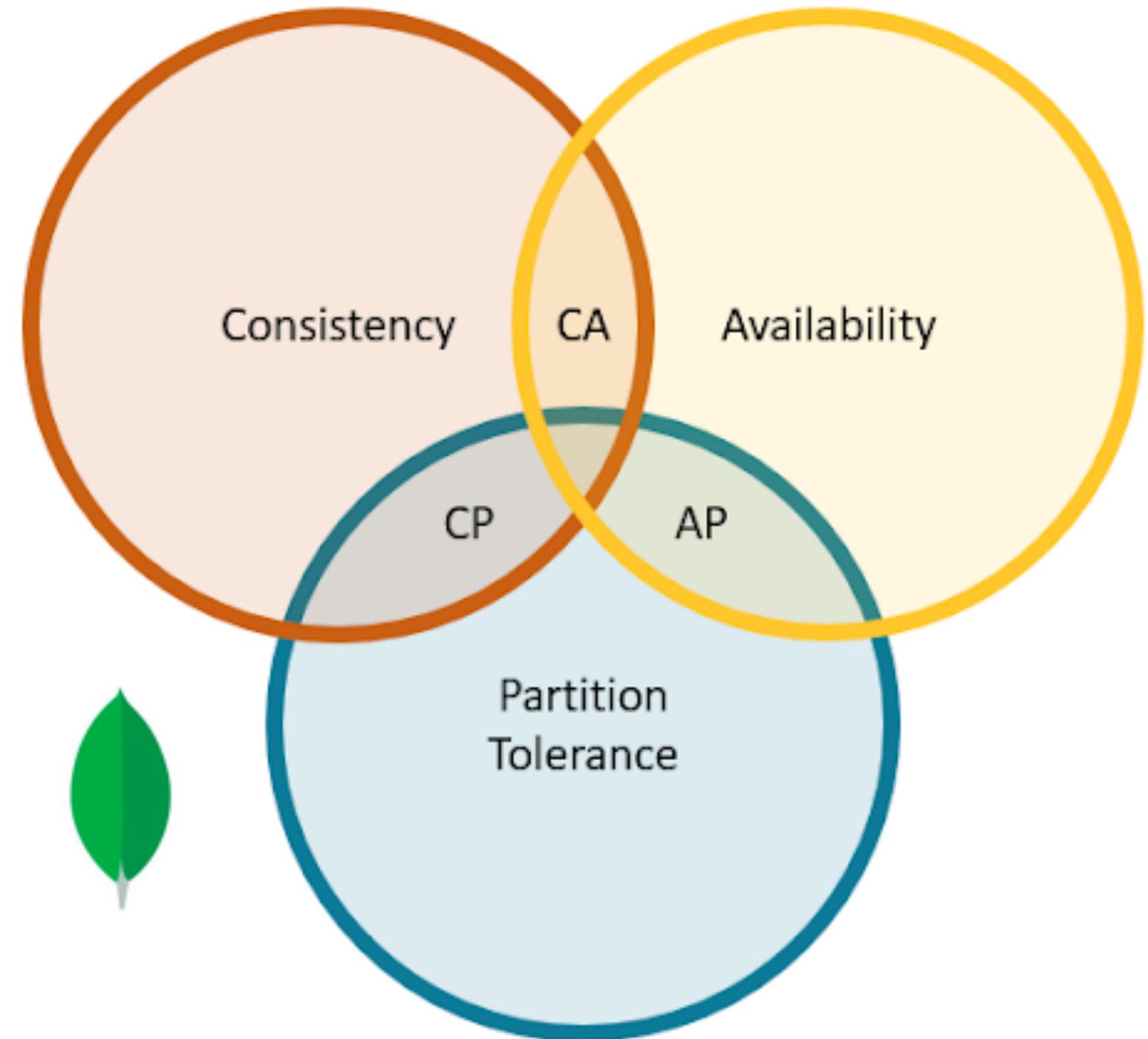
MongoDB

- An In-Memory [[Document Databases]]
- Strong consistency (**C**)
- *Tuneably* available (**A**)



MongoDB

- An In-Memory [[Document Databases]]
- Strong consistency (**C**)
- *Tuneably* available (**A**)
- Horizontal Scalable (**P**)



What to remember

What to remember

- Data Model: JSON

What to remember

- Data Model: JSON
- Storage: BSON

What to remember

- Data Model: JSON
- Storage: BSON
- Operations: iterations

What to remember

- Data Model: JSON
- Storage: BSON
- Operations: iterations
- Replication: Replica Sets

What to remember

- Data Model: JSON
- Storage: BSON
- Operations: iterations
- Replication: Replica Sets
- Partitioning: Sharding

What to remember

- Data Model: JSON
- Storage: BSON
- Operations: iterations
- Replication: Replica Sets
- Partitioning: Sharding
- Architecture

What to remember

- Data Model: JSON
- Storage: BSON
- Operations: iterations
- Replication: Replica Sets
- Partitioning: Sharding
- Architecture
 - Mongod

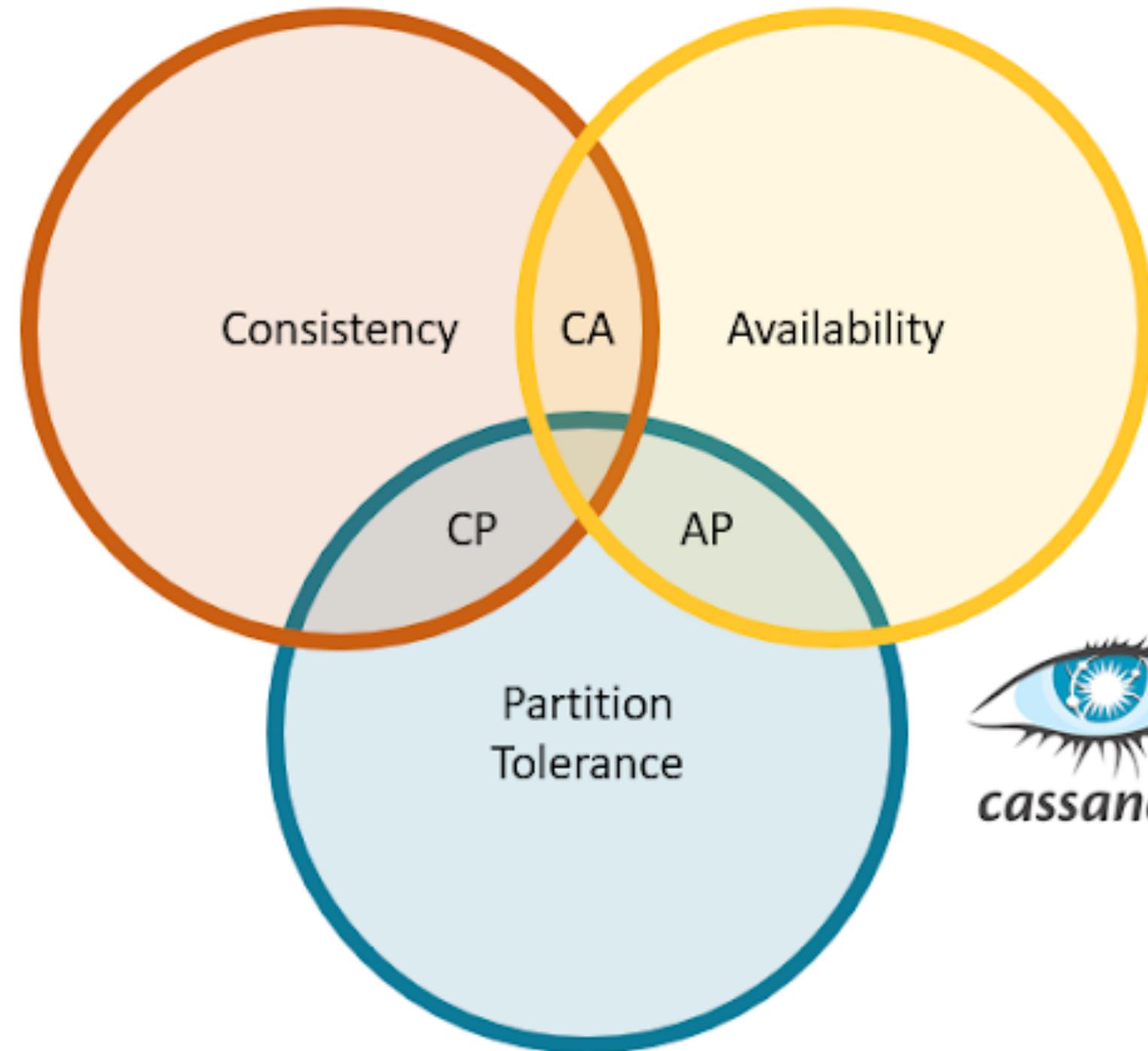
What to remember

- Data Model: JSON
- Storage: BSON
- Operations: iterations
- Replication: Replica Sets
- Partitioning: Sharding
- Architecture
 - Mongod
 - Mongos

What to remember

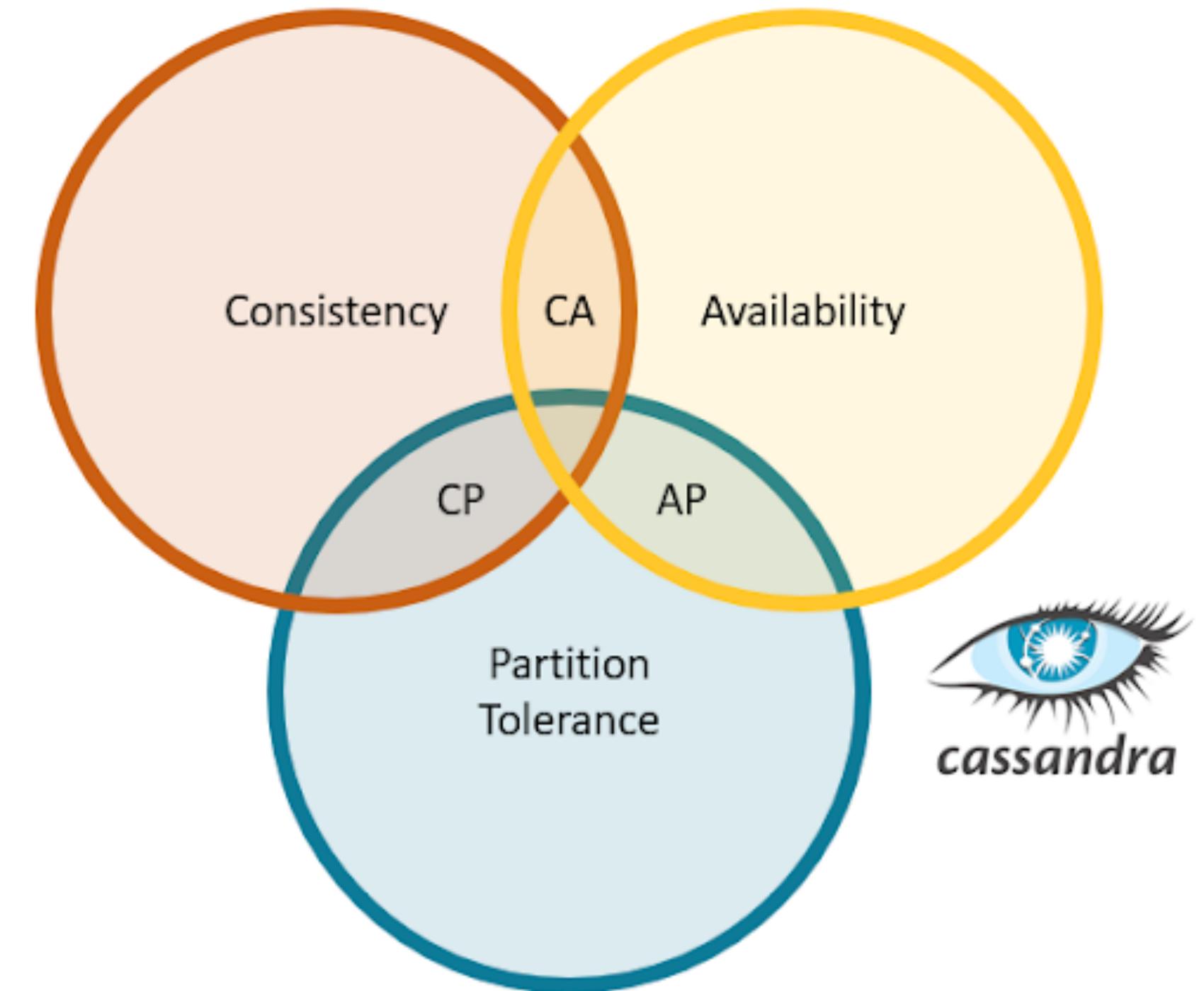
- Data Model: JSON
- Storage: BSON
- Operations: iterations
- Replication: Replica Sets
- Partitioning: Sharding
- Architecture
 - Mongod
 - Mongos
 - Config Server

Cassandra



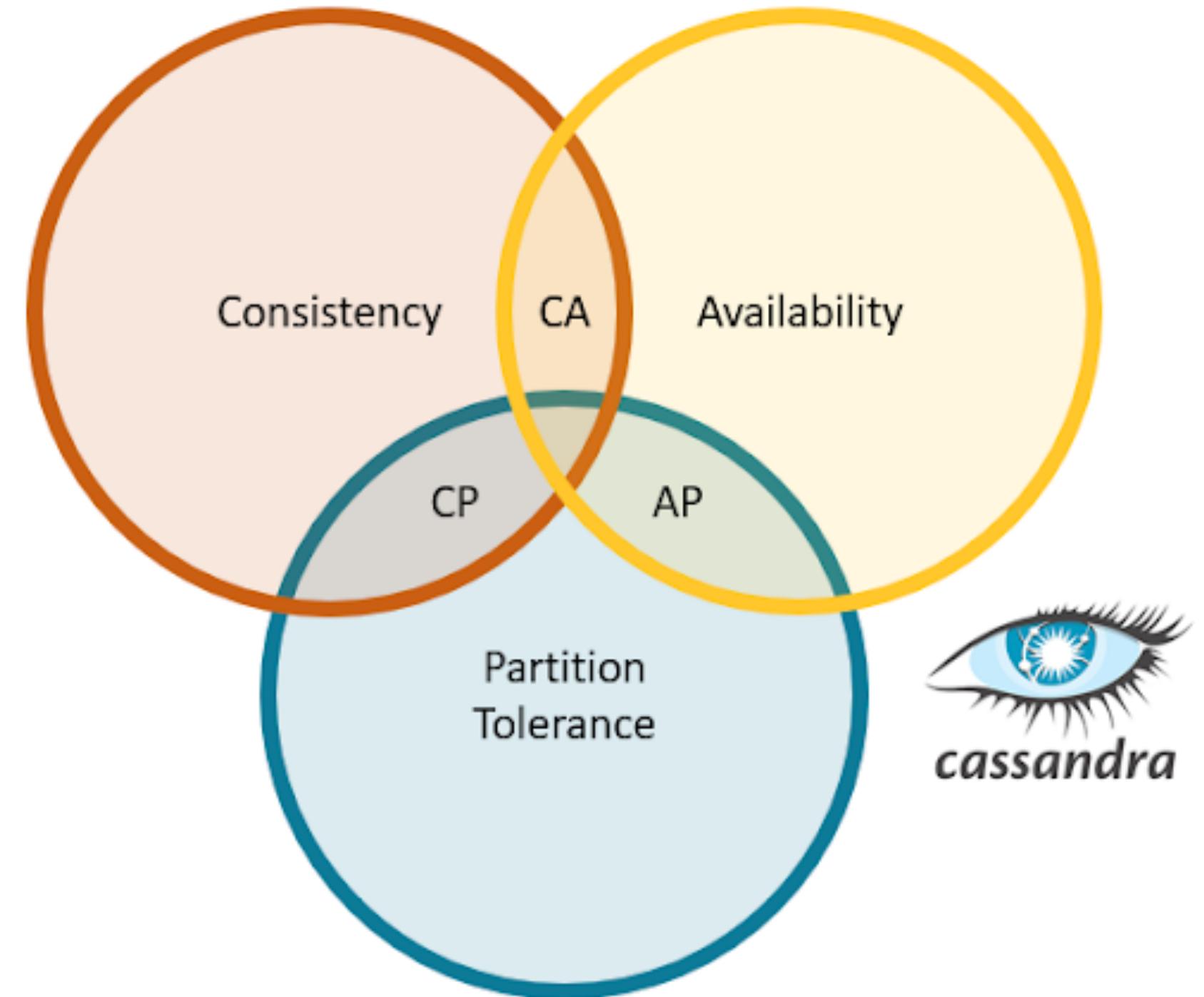
Cassandra

- A Wide [[Column Oriented Database]]



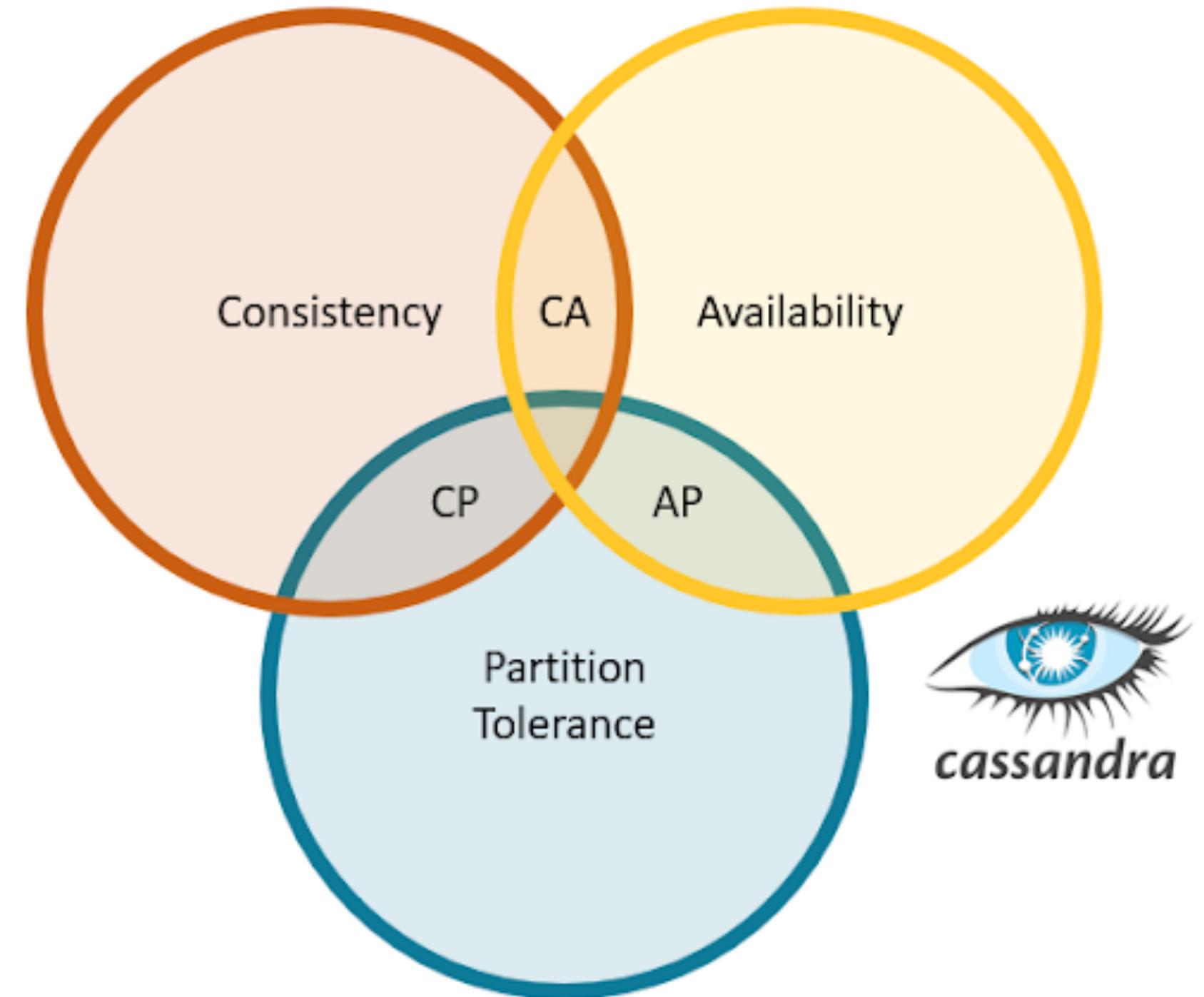
Cassandra

- A Wide [[Column Oriented Database]]
- *tuneably consistent (C)*



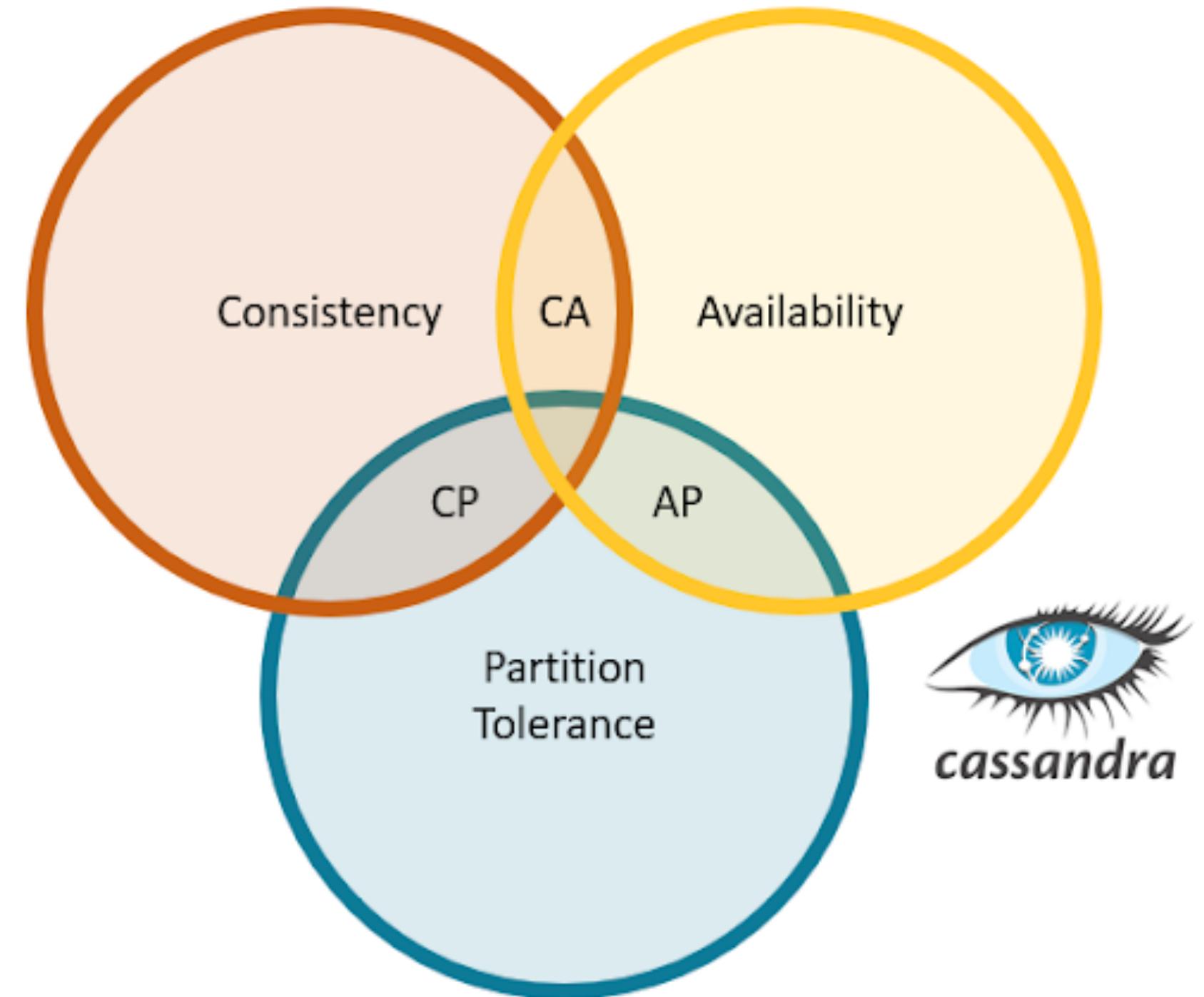
Cassandra

- A Wide [[Column Oriented Database]]
- *tuneably* consistent (ϵ)
- very fast in writes



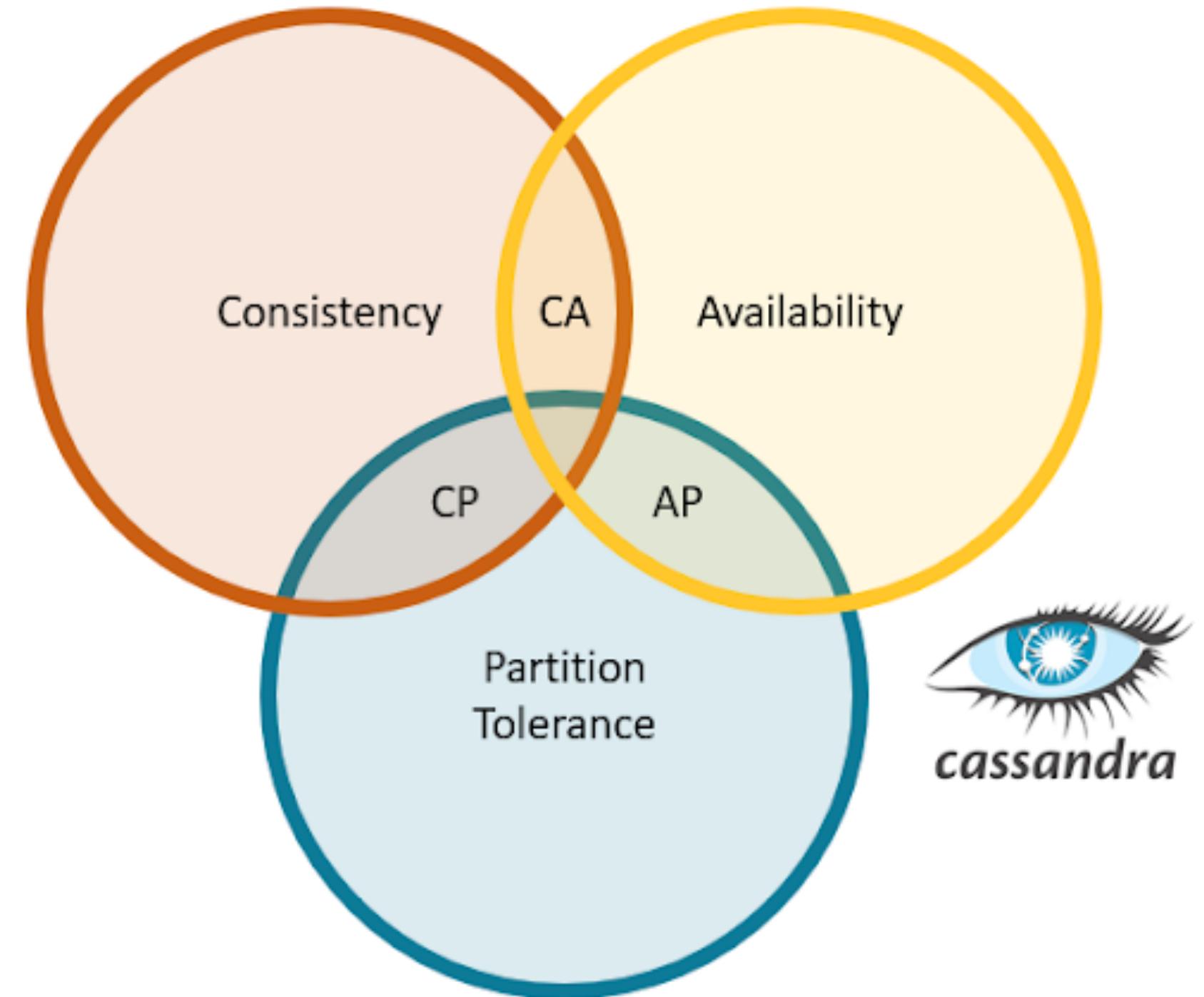
Cassandra

- A Wide [[Column Oriented Database]]
- *tuneably* consistent (ϵ)
- very fast in writes
- highly avaeng



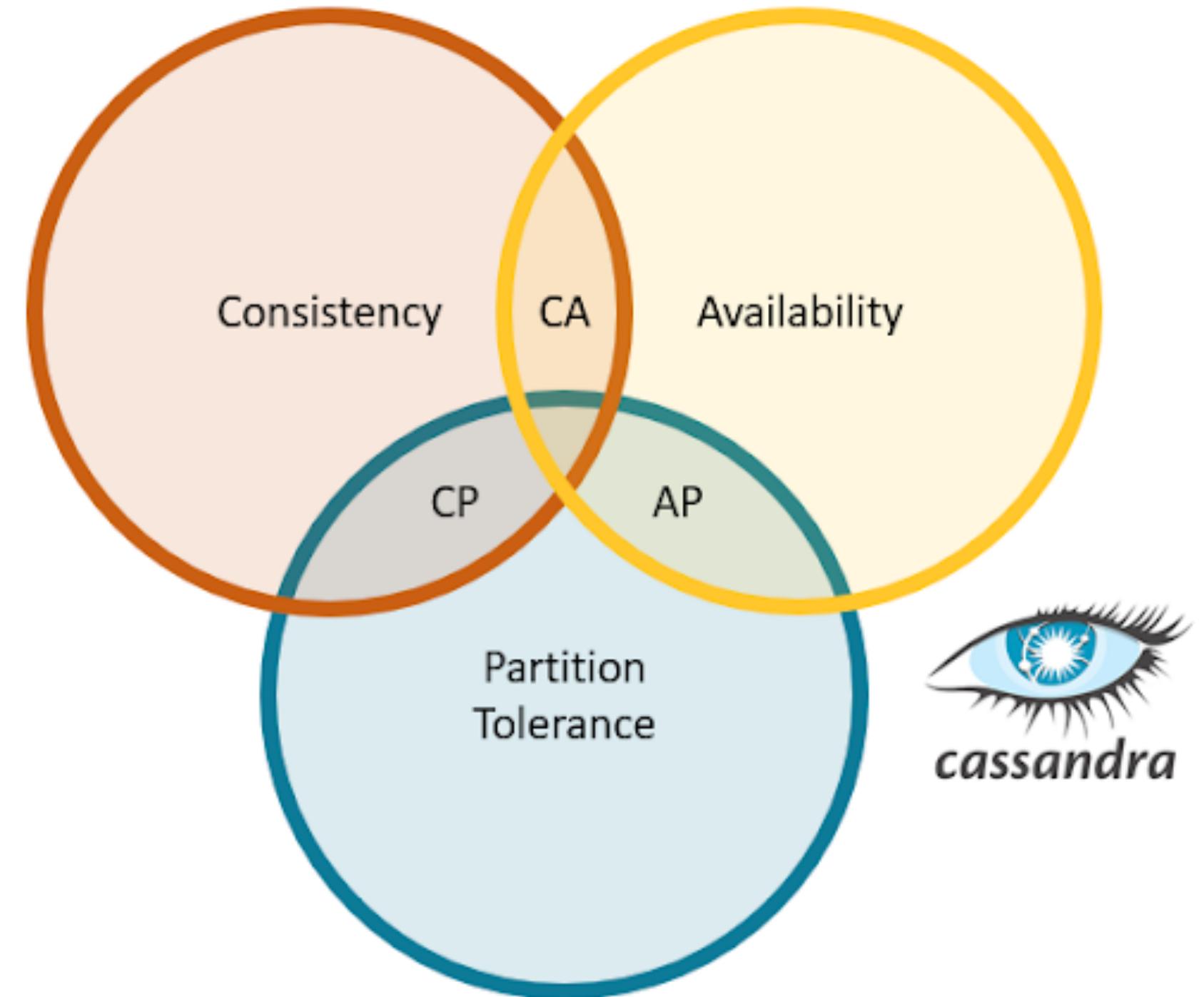
Cassandra

- A Wide [[Column Oriented Database]]
- *tuneably* consistent (ϵ)
- very fast in writes
- highly avaeng
- ailable (A)



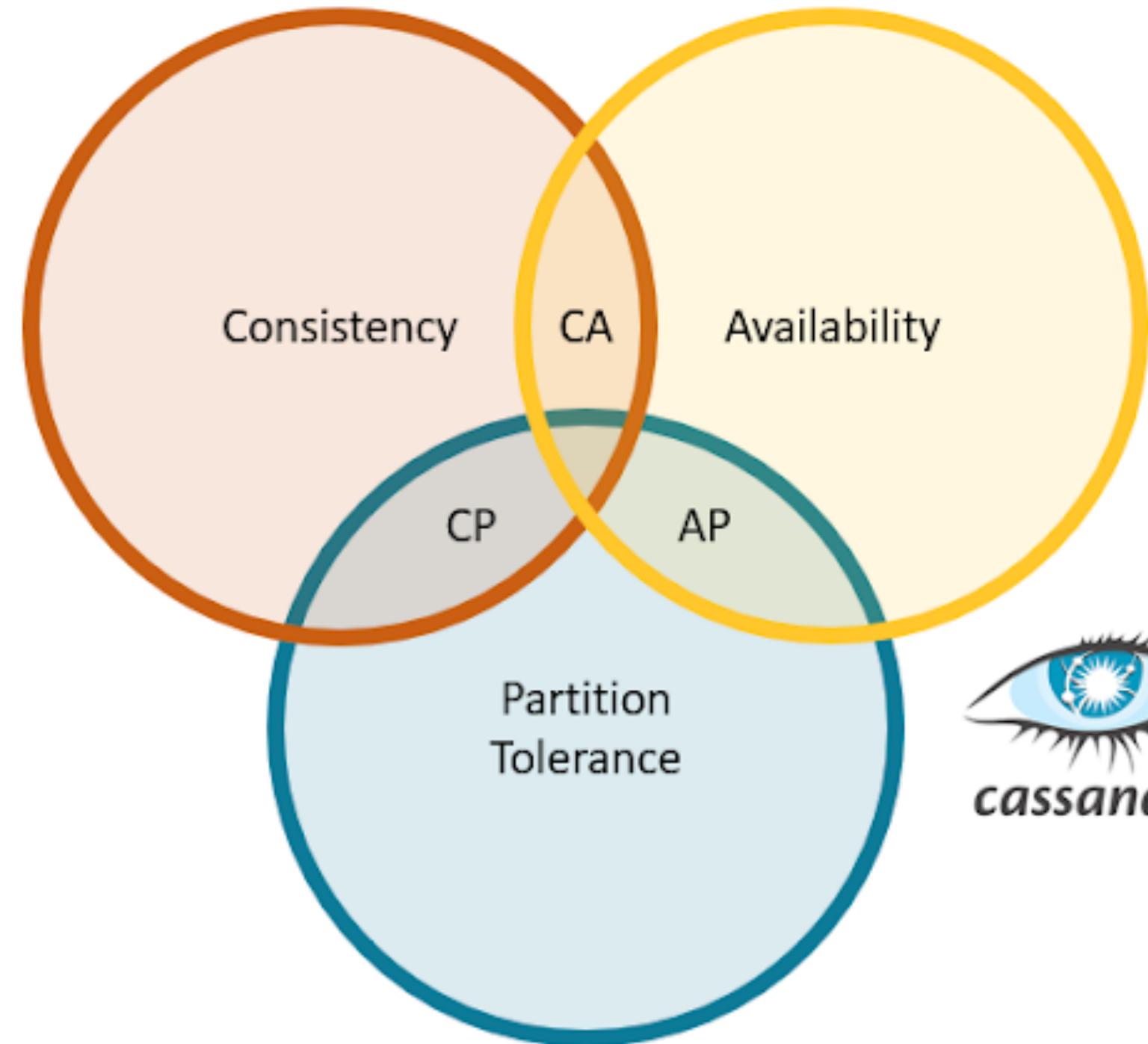
Cassandra

- A Wide [[Column Oriented Database]]
- *tuneably* consistent (**C**)
- very fast in writes
- highly avaeng
- ailable (**A**)
- fault tolerant (**P**)



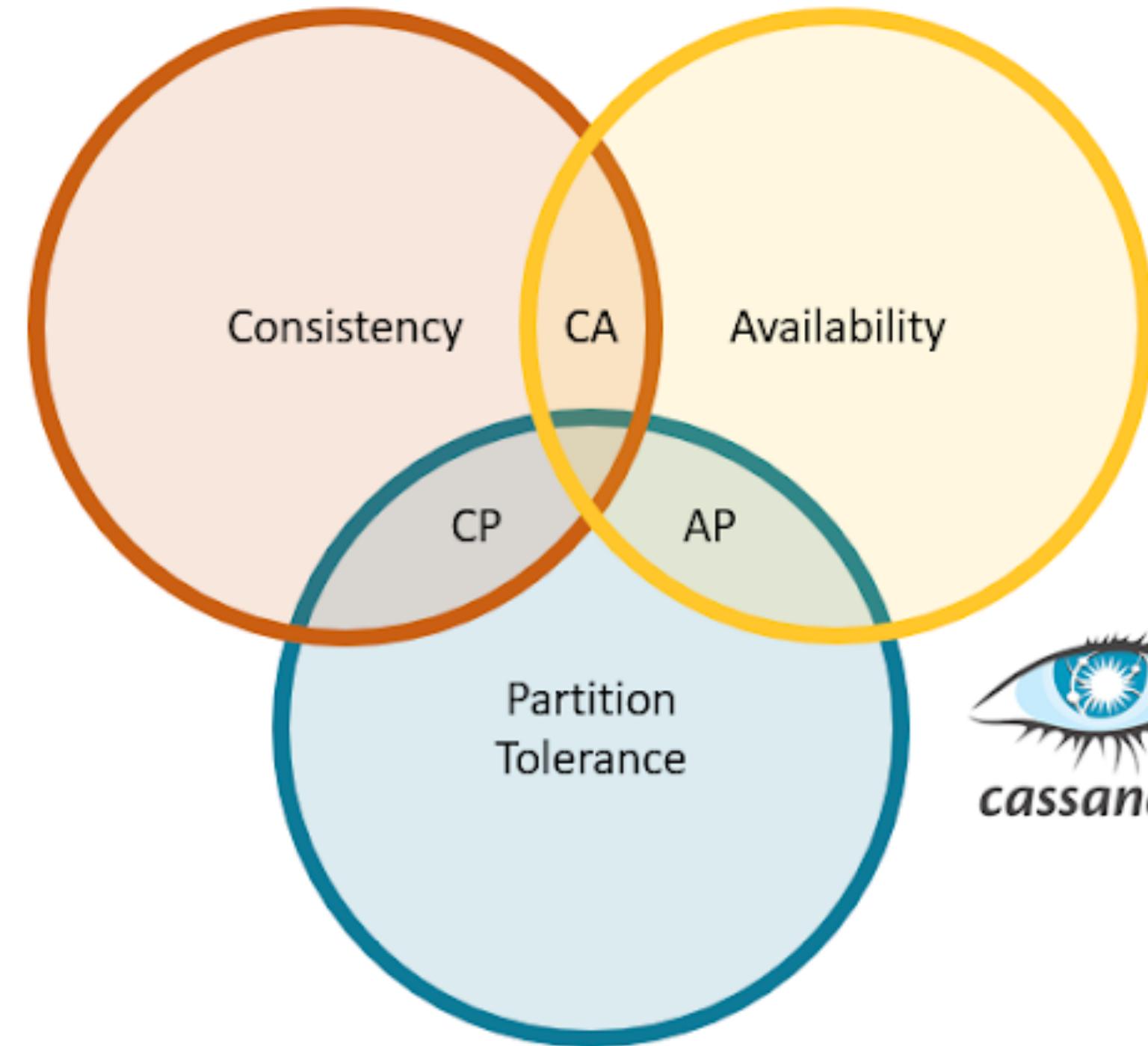
Cassandra

- A Wide [[Column Oriented Database]]
- *tuneably* consistent (**C**)
- very fast in writes
- highly avaeng
- ailable (**A**)
- fault tolerant (**P**)
- linearly scalable, elastic scalability



Cassandra

- A Wide [[Column Oriented Database]]
- *tuneably* consistent (**C**)
- very fast in writes
- highly avaeng
- ailable (**A**)
- fault tolerant (**P**)
- linearly scalable, elastic scalability
- Cassandra is very good at writes, okay with reads.



What to remember

What to remember

- Data Model: KeySpace, Rows, Column Families

What to remember

- Data Model: KeySpace, Rows, Column Families
- Storage: Commit Log, SSTables, Bloom Filters

What to remember

- Data Model: KeySpace, Rows, Column Families
- Storage: Commit Log, SSTables, Bloom Filters
- Operations: ~~SQL~~ (CQL!)

What to remember

- Data Model: KeySpace, Rows, Column Families
- Storage: Commit Log, SSTables, Bloom Filters
- Operations: ~~SQL~~ (CQL!)
 - Different Consistency Levels for reads and writes

What to remember

- Data Model: KeySpace, Rows, Column Families
- Storage: Commit Log, SSTables, Bloom Filters
- Operations: ~~SQL~~ (CQL!)
 - Different Consistency Levels for reads and writes
- Replication: Replication factor

What to remember

- Data Model: KeySpace, Rows, Column Families
- Storage: Commit Log, SSTables, Bloom Filters
- Operations: ~~SQL~~ (CQL!)
 - Different Consistency Levels for reads and writes
- Replication: Replication factor
 - Simple vs Network Topology

What to remember

- Data Model: KeySpace, Rows, Column Families
- Storage: Commit Log, SSTables, Bloom Filters
- Operations: ~~SQL~~ (CQL!)
 - Different Consistency Levels for reads and writes
- Replication: Replication factor
 - Simple vs Network Topology
- Partitioning: Order Preserving

What to remember

- Data Model: KeySpace, Rows, Column Families
- Storage: Commit Log, SSTables, Bloom Filters
- Operations: ~~SQL~~ (CQL!)
 - Different Consistency Levels for reads and writes
- Replication: Replication factor
 - Simple vs Network Topology
- Partitioning: Order Preserving
- Architecture: Gossip Protocol

GraphDBs (Neo4J)

GraphDBs (Neo4J)

- Back to Centralized Architecture

GraphDBs (Neo4J)

- Back to Centralized Architecture
 - Relationship First

GraphDBs (Neo4J)

- Back to Centralized Architecture
 - Relationship First
 - Storage

GraphDBs (Neo4J)

- Back to Centralized Architecture
 - Relationship First
 - Storage
 - Native vs Non Native

GraphDBs (Neo4J)

- Back to Centralized Architecture
 - Relationship First
 - Storage
 - Native vs Non Native
 - Graph Pattern Matching

GraphDBs (Neo4J)

- Back to Centralized Architecture
 - Relationship First
 - Storage
 - Native vs Non Native
 - Graph Pattern Matching
 - Union

GraphDBs (Neo4J)

- Back to Centralized Architecture
 - Relationship First
 - Storage
 - Native vs Non Native
 - Graph Pattern Matching
 - Union
 - Projection

GraphDBs (Neo4J)

- Back to Centralized Architecture
 - Relationship First
 - Storage
 - Native vs Non Native
 - Graph Pattern Matching
 - Union
 - Projection
 - Different

GraphDBs (Neo4J)

- Back to Centralized Architecture
 - Relationship First
 - Storage
 - Native vs Non Native
 - Graph Pattern Matching
 - Union
 - Projection
 - Different
 - Optional

GraphDBs (Neo4J)

- Back to Centralized Architecture
 - Relationship First
 - Storage
 - Native vs Non Native
 - Graph Pattern Matching
 - Union
 - Projection
 - Different
 - Optional
 - Filter

GraphDBs (Neo4J)

- Back to Centralized Architecture
 - Relationship First
 - Storage
 - Native vs Non Native
 - Graph Pattern Matching
 - Union
 - Projection
 - Different
 - Optional
 - Filter
 - Navigational Queries (Paths)

Questions Template

- What are the three levels of data modeling

- What are the three levels of data modeling
 - A) Conceptual, Logical, Physical

- What are the three levels of data modeling
 - A) Conceptual, Logical, Physical
 - B) Perceptual, Latent, Physical

- What are the three levels of data modeling
 - A) Conceptual, Logical, Physical
 - B) Perceptual, Latent, Physical
 - C) Coherent, Logical, Phenomenal

- What are the three levels of data modeling

- What are the three levels of data modeling
 - A) **Conceptual, Logical, Physical**

- What are the three levels of data modeling
 - A) **Conceptual, Logical, Physical**
 - B) Perceptual, Latent, Physical

- What are the three levels of data modeling
 - A) **Conceptual, Logical, Physical**
 - B) Perceptual, Latent, Physical
 - C) Coherent, Logical, Phenomenal

- What is a fact table

- What is a fact table
 - A) a table that contains the descriptive attributes used by BI applications for filtering and grouping the facts

- What is a fact table
 - A) a table that contains the descriptive attributes used by BI applications for filtering and grouping the facts
 - B) a table with a one-to-one relationship to a measurement event as described by the fact table's grain.

- What is a fact table
 - A) a table that contains the descriptive attributes used by BI applications for filtering and grouping the facts
 - B) a table with a one-to-one relationship to a measurement event as described by the fact table's grain.
 - C) a table that contains the numeric measures produced by an operational measurement event in the real world.

- What is a fact table

- What is a fact table
 - A) a table that contains the descriptive attributes used by BI applications for filtering and grouping the facts

- What is a fact table
 - A) a table that contains the descriptive attributes used by BI applications for filtering and grouping the facts
 - B) a table with a one-to-one relationship to a measurement event as described by the fact table's grain.

- What is a fact table
 - A) a table that contains the descriptive attributes used by BI applications for filtering and grouping the facts
 - B) a table with a one-to-one relationship to a measurement event as described by the fact table's grain.
 - C) a table that contains the numeric measures produced by an operational measurement event in the real world.

What of the following are the valid ADT to represent graphs

- A) Edge List
- B) Adjacency List
- C) Adjacency Incident
- D) Edge Matrix

What of the following are the motivations behind the NoSQL movement

- need for greater scalability than relational databases can't offer
- need for specialized query operations
- need for a more dynamic and expressive data model
- the Object-Relational Mismatch

What of the following are the motivations behind the NoSQL movement

- need for greater scalability than relational databases can't offer
- need for specialized query operations
- need for a more dynamic and expressive data model **
- **the Object-Relational Mismatch

What of the following are the valid ADT to represent graphs

- A) **Edge List**
- B) **Adjacency List**
- C) Adjacency Incident
- D) Edge Matrix