Foundation of Data Engineering

MCF Riccardo Tommasini

http://rictomm.me

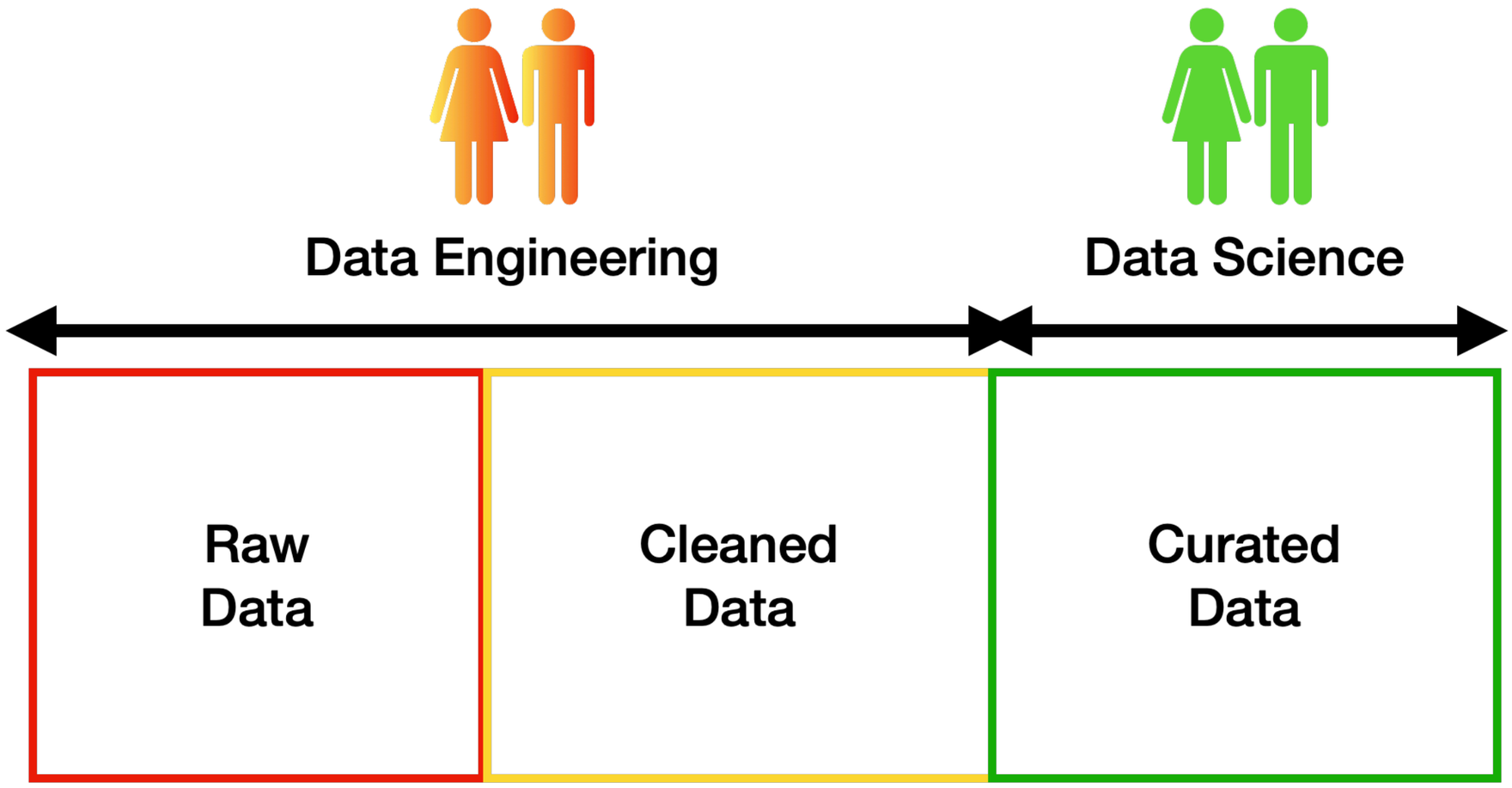
riccardo.tommasini@insa-lyon.fr

Recap

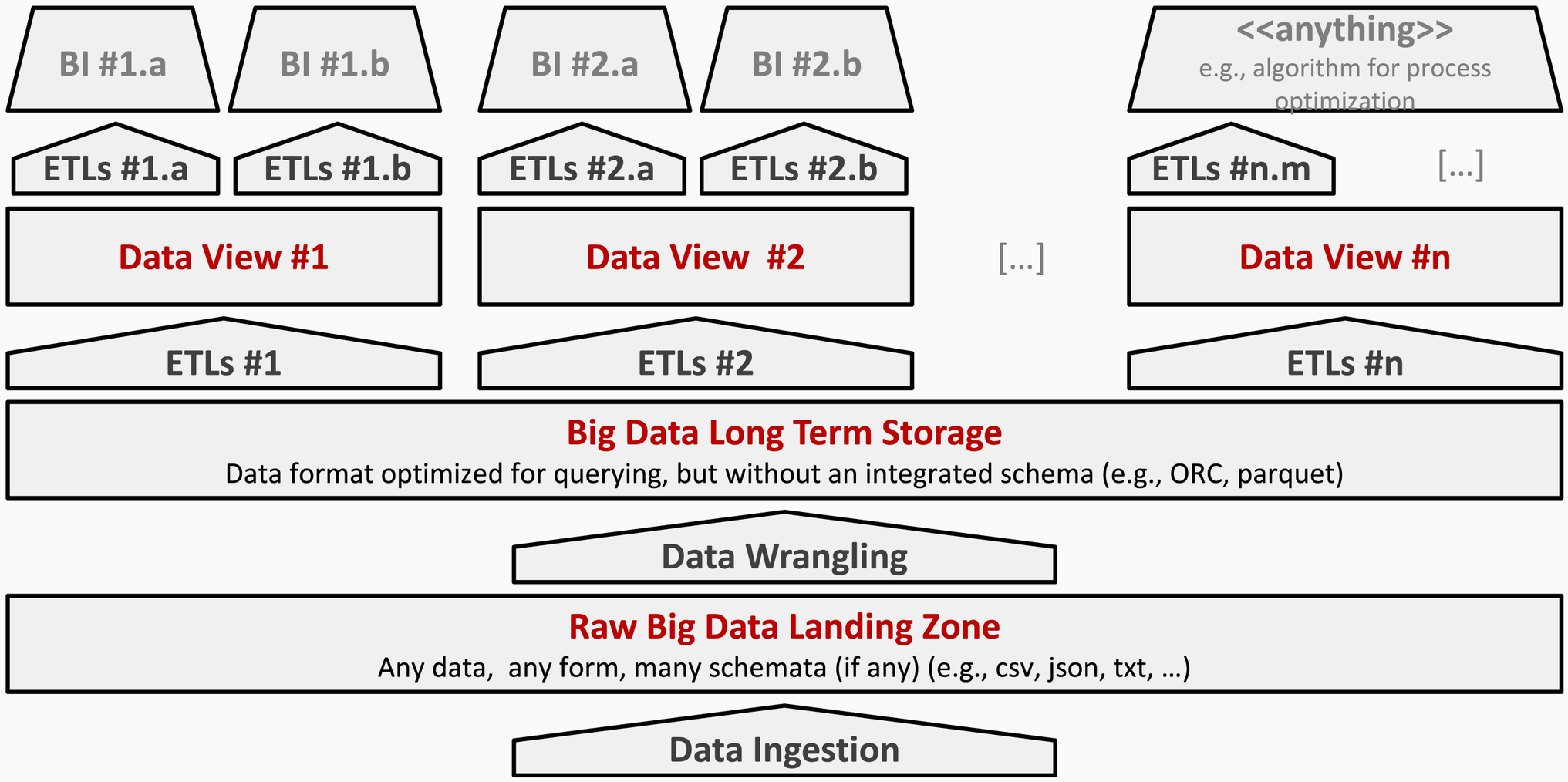
- The different roles of data engineer and scientist
- Data quality identifies the "zones"
- How do traverse the conceptual zones: pipelines!

- However, the data pipeline has been quite an abstract concept so far
- We still miss the elements that implements the pipeline (Airflow)
- We did not discuss concrete problems like durability and distribution

A Conceptual View of a Data Pipeline

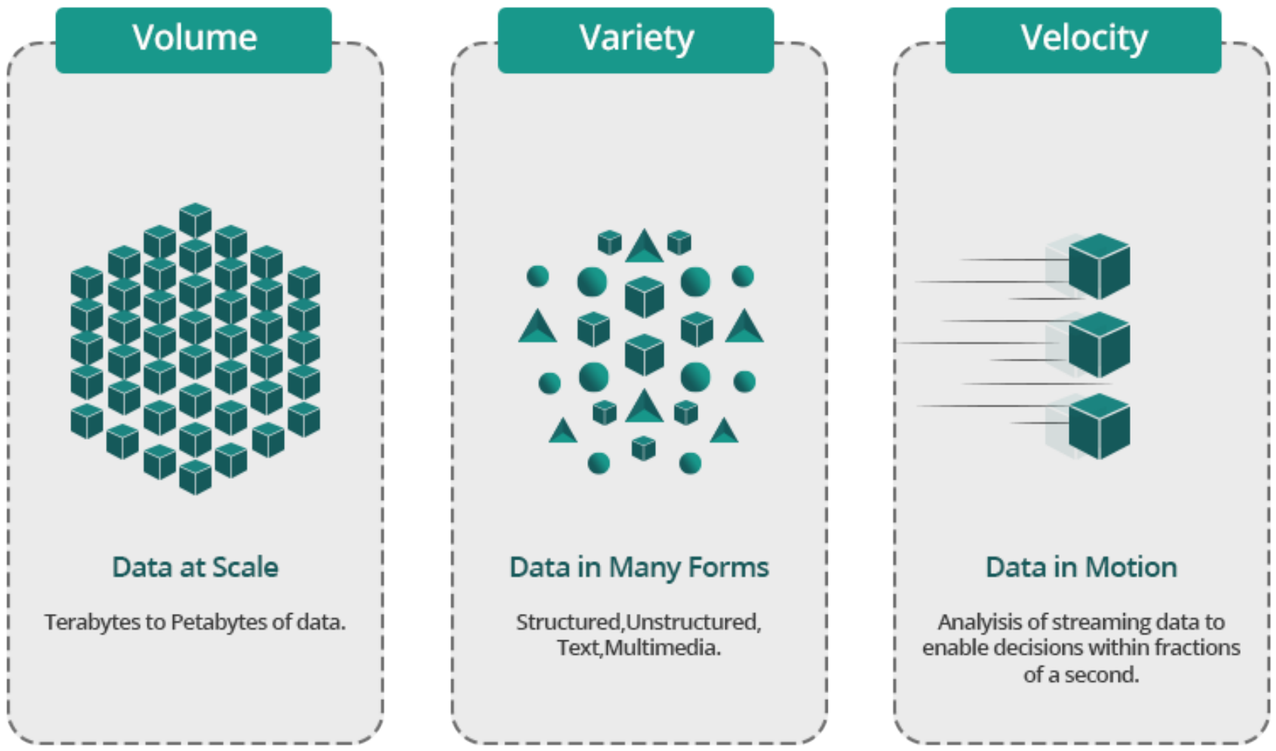


Logical architecture of a data engineering pipeline



Towards a Physical View

- Big data have an essential role in today's pipeline design
- As we said, this is **not** just about the size!
  - **Volume**: demands scalability of storage
  - **Variety**: calls for flexibility of schema
  - **Velocity**: requires continuous processing



- Once again the big data challenges impact the design of our pipelines
- The are all relevant at many levels, but volume is the one that caused most of the changes
- we need to relax some aspects of the data systems

Towards a Physical View

Before digging into the details of the physical view, we need to unveil two premises

- A Distributed System Premise: Big Data imply data partitioning
- A Data System Premise: Big Data dispute data modelling as it was

Data Partitioning

breaking a large database down into smaller ones

The main reason for wanting to partition data is scalability[13]

[13] Designing Data-Intensive Applications

For very large datasets, or very high query throughput, that is not sufficient

- Different partitions can be placed on different nodes in a shared-nothing cluster
- Queries that operate on a single partition can be independently executed. Thus, throughput can be scaled by adding more nodes.

What to know

- If some partitions have more data or queries than others the partitioning is **skewed**
- A partition with disproportionately high load is called a **hot spot**
- For reaching maximum scalability (linear) partitions should be balanced

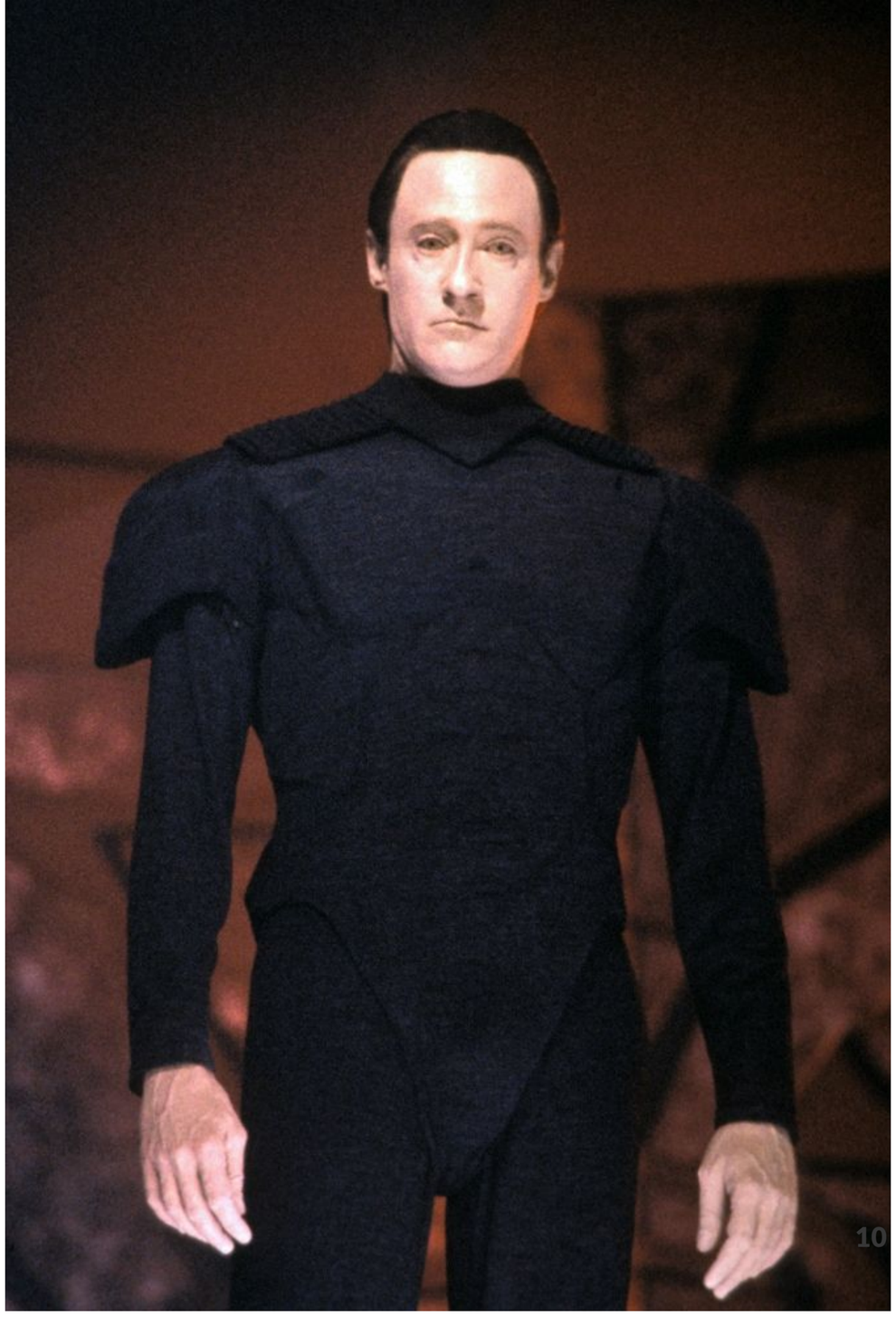
Let's consider some partitioning strategies, for simplicity we consider Key,Value data.

Partitioning Strategies

- **Round-robin** randomly assigns new keys to the partitions.
  - Ensures an even distribution of tuples across nodes;
- **Range partitioning** assigns a contiguous key range to each node.
  - Not necessarily balanced, because data may not be evenly distributed
- **Hash partitioning** uses a hash function to determine the target partition. - If the hash function returns i, then the tuple is placed

Data Modeling

It is the process of defining the structure of the data for the purpose of communicating[11] or to develop an information systems[12].



[11] between functional and technical people to show data needed for business processes

[12] between components of the information system, how data is stored and accessed.

What is a data model?

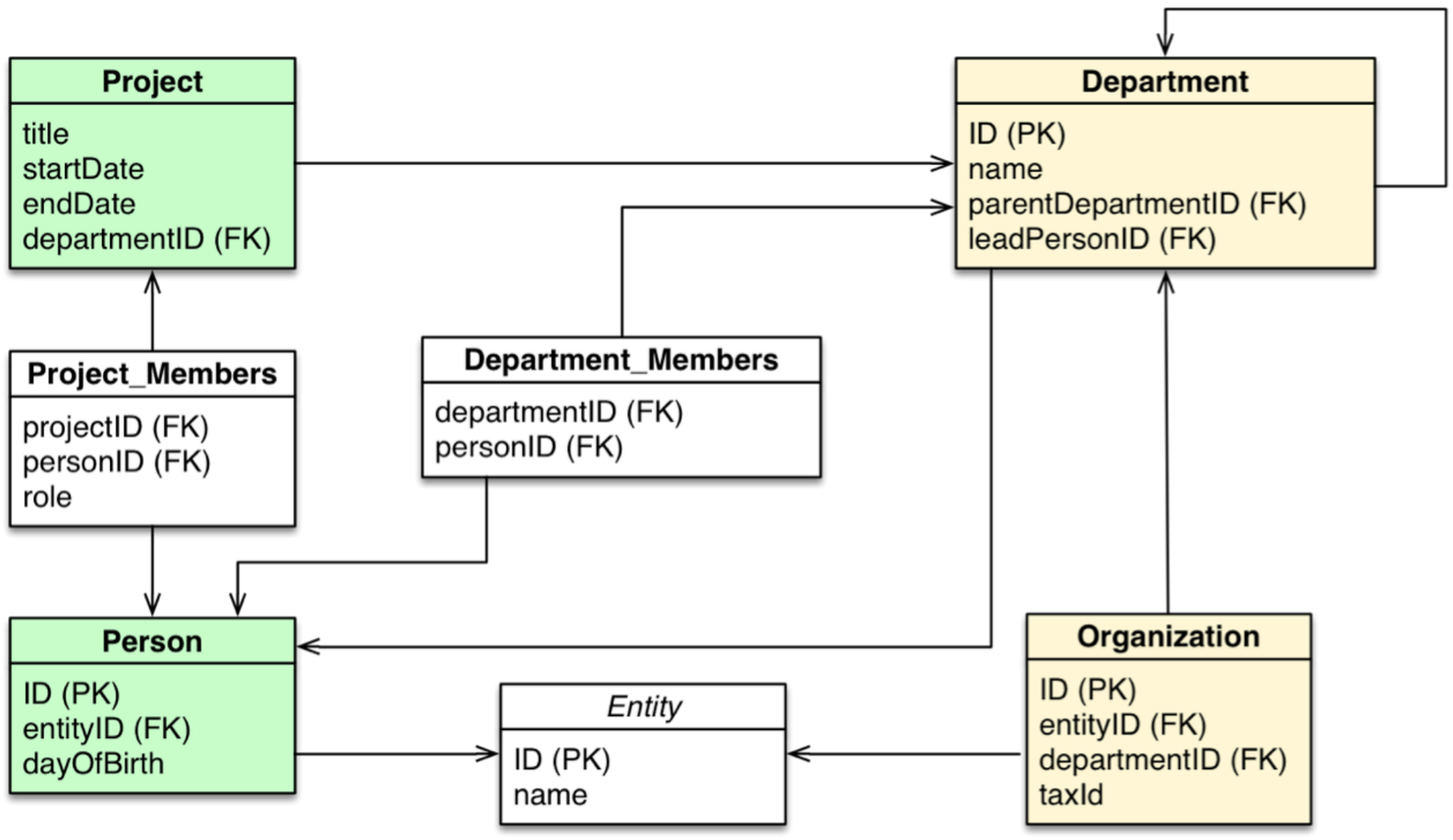
- A data model represents the structure and the integrity of the data elements of a (single) applications 2
- Data models provide a framework for data to be used within information systems by giving specific definitions and formats.
- The literature of data management is rich of data models that aim at providing increased expressiveness to the modeller and capturing a richer set of semantics.

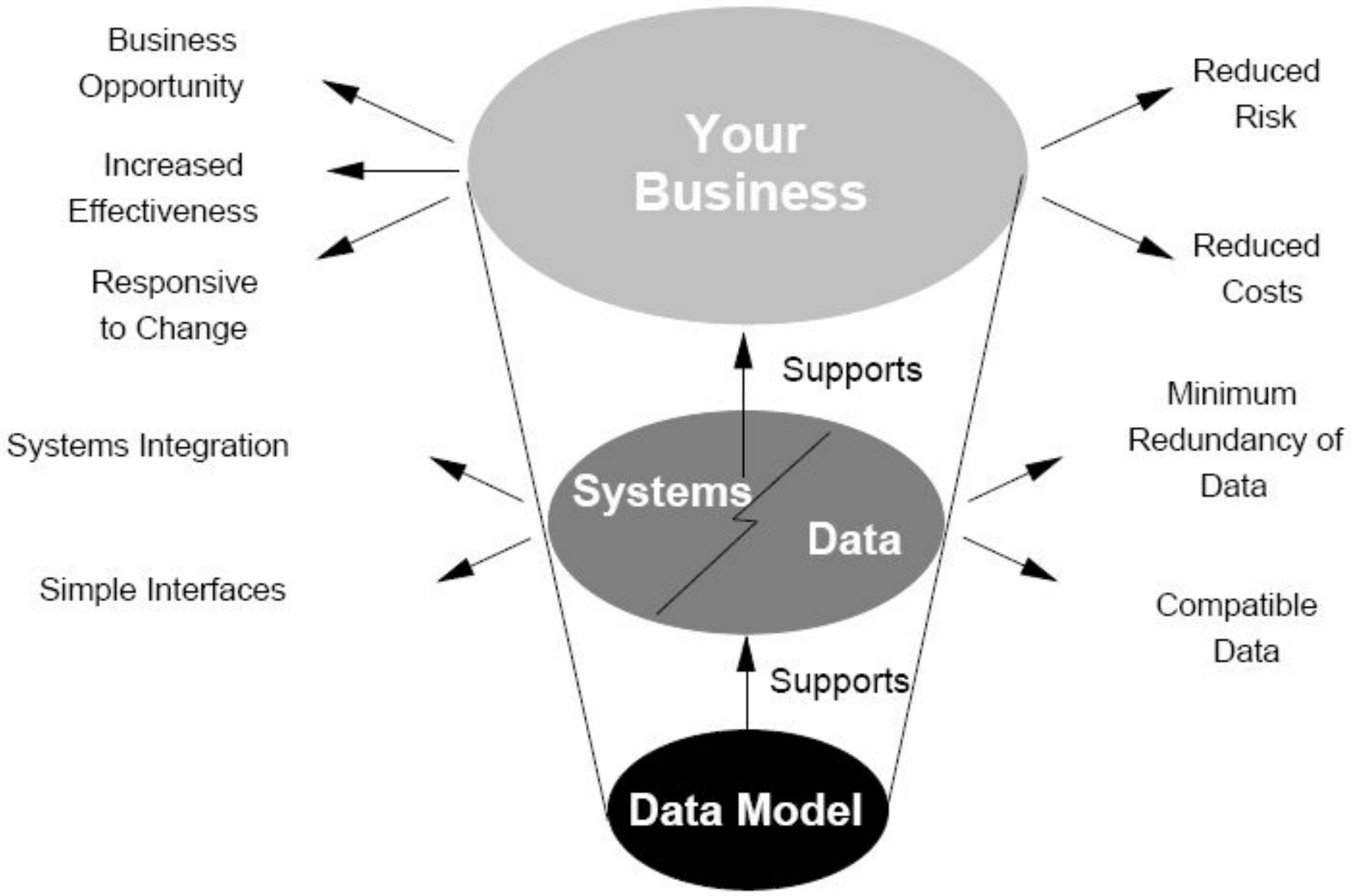
Data models are perhaps the most important part of developing software. They have such a profound effect not only on how the software is written, but also on how we think about the problem that we are solving[13].

— Martin Kleppmann

[13] Designing Data-Intensive Applications

Any Example?


Project
title
startDate
endDate
departmentID (FK)
Department
ID (PK)
name
parentDepartmentID (FK)
leadPersonID (FK)
Project_Members
projectID (FK)
personID (FK)
role
Department_Members
departmentID (FK)
personID (FK)
Person
ID (PK)
entityID (FK)
dayOfBirth
Entity
ID (PK)
name
Organization
ID (PK)
entityID (FK)
departmentID (FK)
taxId

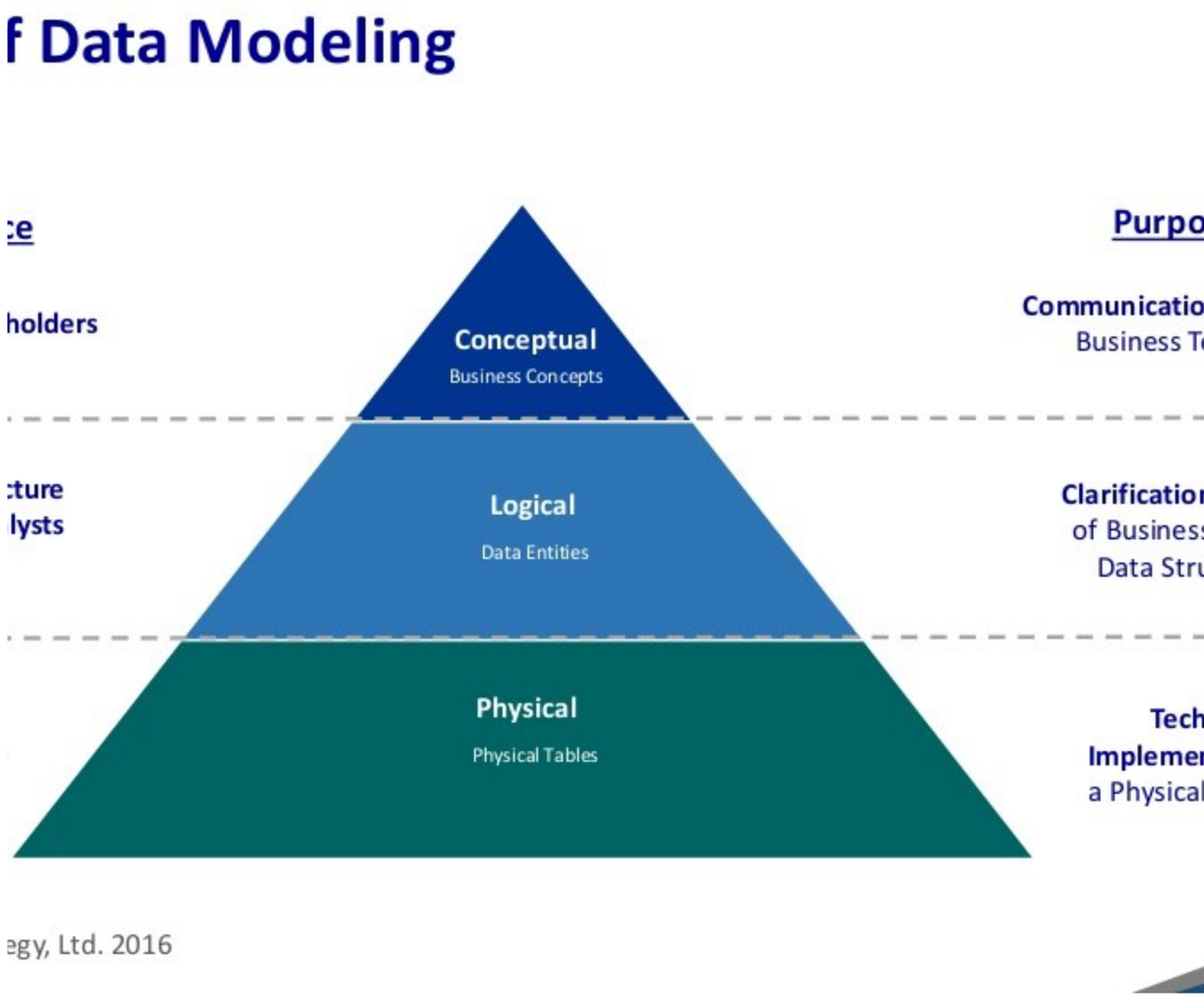

Business Opportunity
Increased Effectiveness
Responsive to Change
Systems Integration
Simple Interfaces
Your Business
Supports
Systems
Data
Supports
Data Model
Reduced Risk
Reduced Costs
Minimum Redundancy of Data
Compatible Data

Level of Data Modeling

Conceptual: The data model defines *WHAT* the system contains.

Logical: Defines *HOW* the system should be implemented regardless of the DBMS.

Physical: This Data Model describes *HOW* the information system will be implemented using a specific technology [14].



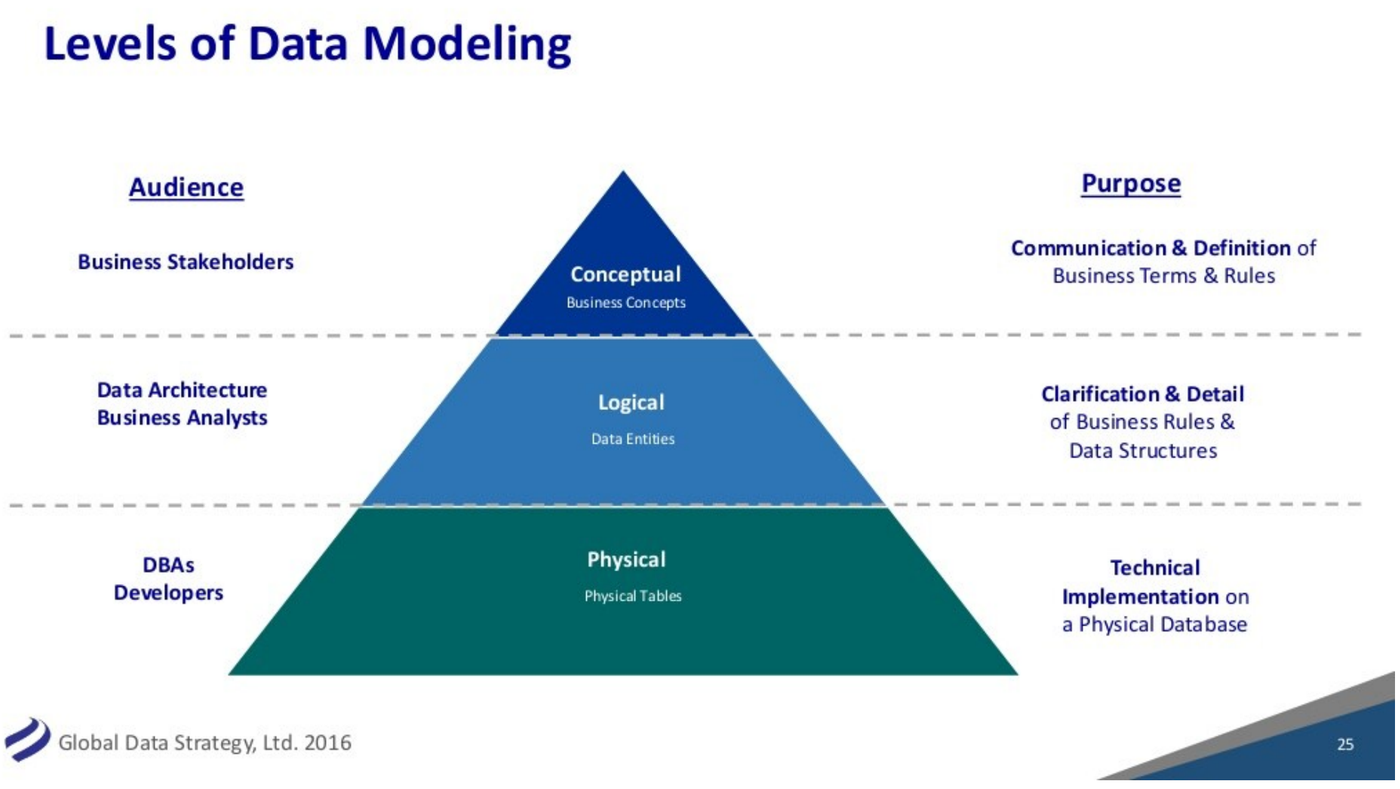
[14] physical

Conceptual model is typically created by Business stakeholders. The purpose is to organize, scope and define business concepts and rules. Definitions are most important this level.

Logical model is typically created by Data Architects. The purpose is to developed technical map of rules and data structures. Business rules, relationships, attribute become visible. Conceptual definitions become metadata.

Physical model is typically created by DBA and developers. The purpose is actual implementation of the database. Trade-offs are explored by in terms of data structures and algorithms.

A Closer Look[15]



[15] slides & video by Donna Burbank

The variety of data available today encourages the design and development of dedicated data models and query languages that can improve both BI as well as the engineering process itself.


We need help from Rudyard Kipling
Copyright © 2012, Essential Strategies, Inc.
24/34

Conceptual

- Semantic Model (divergent)
  - Describes an enterprise in terms of the language it uses (the jargon).
  - It also tracks inconsistencies, i.e., semantic conflicts
- Architectural Model (convergent)
  - More fundamental, abstract categories across enterprise

Logical

Already bound to a technology, it typically refers already to implementation details

- Relational
- Hierarchical
- Key-Value
- Object-Oriented
- Graph

Since it has a physical bias, you might be tempted to confuse this with the physical model, but this is wrong.

Physical

The physical level describes how data are **Stored** on a device.

- Data formats
- Distribution
- Indexes
- Data Partitions
- Data Replications

...an you are in the Big Data World

Towards a Physical View

Before digging into the details of the physical view, we need to unveil two premises

- A Distributed System Premise: CAP Theorem
- A Data System Premise: NoSQL%20SQL.md)

CAP Theorem (Brewer's Theorem)

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

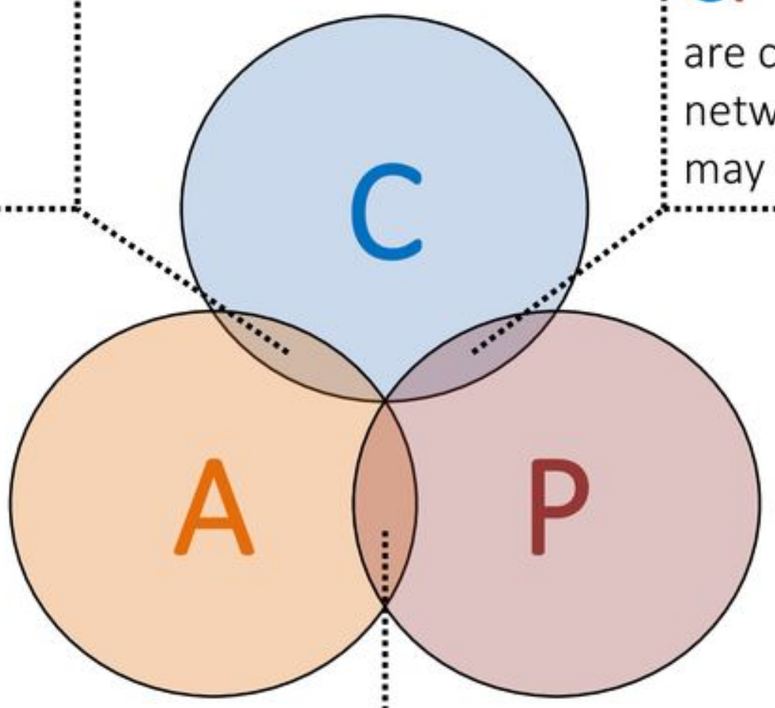
- **Consistency**: all nodes see the same data at the same time
- **Availability**: Node failures do not prevent other survivors from continuing to operate (a guarantee that every request receives a response whether it succeeded or failed)
- **Partition tolerance**: the system continues to operate despite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

A distributed system can satisfy any two of these guarantees at the same time but not all three.

CAP Systems

CA: Guarantees to give a correct response but only while network works fine (*Centralised / Traditional*)

CP: Guarantees responses are correct even if there are network failures, but response may fail (*Weak availability*)



(No intersection)

AP: Always provides a "best-effort" response even in presence of network failures (*Eventual consistency*)

The network is not reliable

In a distributed system, **a network (of networks)** failures can, and will, occur.

Since We cannot neglect Partition Tolerance the remaining option is choosing between **Consistency** and **Availability**.

We cannot neglect Partition Tolerance

Not necessarily in a mutually exclusive manner:

- CP: A partitioned node returns
  - the correct value
  - a timeout error or an error, otherwise
- AP: A partitioned node returns the most recent version of the data, which could be stale.

Implications of CAP Theorem

- change the transactionality gurantees
- redesign the data workflow ()
- reimagine the data processing systems (noSQL)

The Advent of NoSQL

Google, Amazon, Facebook, and DARPA all recognised that when you scale systems large enough, you can never put enough iron in one place to get the job done (and you wouldn't want to, to prevent a single point of failure).

Once you accept that you have a distributed system, you need to give up consistency or availability, which the fundamental transactionality of traditional RDBMSs cannot abide.
--Cedric Beust

The name "NoSQL" is unfortunate, since it doesn't actually refer to any particular technology—it was originally intended simply as a catchy Twitter hashtag for a meetup on open source, distributed, non-relational databases in 2009 Cf Pramod J. Sadalage and Martin Fowler: NoSQL Distilled. Addison-Wesley, August 2012. ISBN: 978-0-321-82662-6

The Reasons Behind

- **Queryability**: need for specialised query operations that are not well supported by the relational model
- **Schemaless**: desire for a more dynamic and expressive data model than relational
- **Flexibility**: need to accomodate the "schema on read" phylosophy

– **Big Data**: need for greater scalability than relational databases can easily achieve *in write*

– **Open Source:** a widespread preference for free and open source software

Object-Relational Mismatch

Most application development today is done in **object-oriented** programming languages

An **awkward translation** layer is required between the **objects** in the application code and the database model of **tables**, **rows**, and **columns**

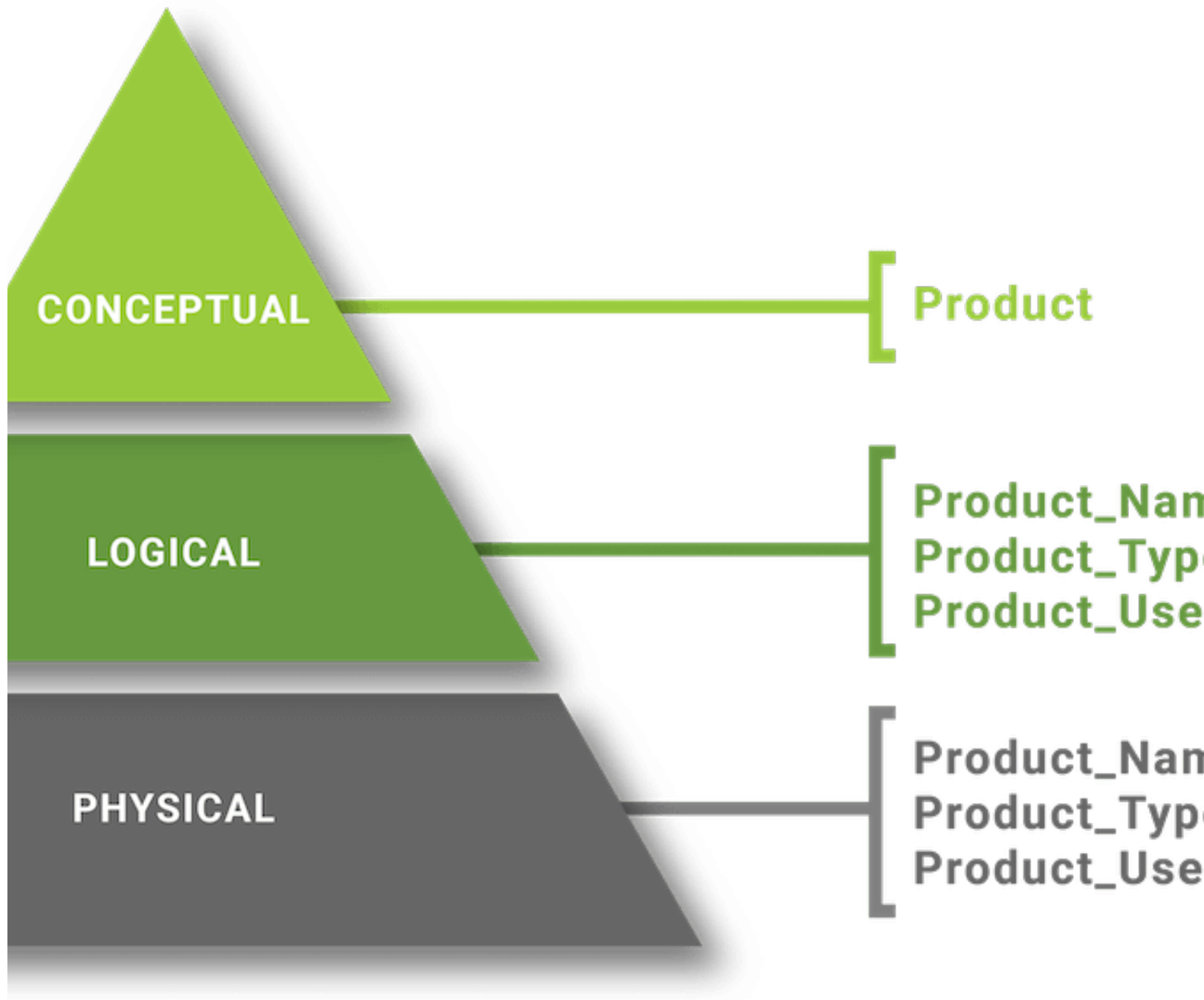
Object-relational mapping (**ORM**) frameworks like **Hibernate** try to mild the mismatch, but they **can’t completely hide** the differences

the idea of NOSQL actually originates in the late 60s together with the raise of the raise of object-oriented languages, but become popular later.

Shall we rethink the three-layered modelling for Big Data?

Designing NoSQL Data Structures

- NoSQL data structures driven by application design.
  - Need to take into account necessary CRUD operations
- To embed or not to imbed. That is the question!
  - Rule of thumb is to imbed whenever possible.
- No modelling standards or CASE[case] tools!



[case] computer aided software engineering

Data Modeling for Big Data

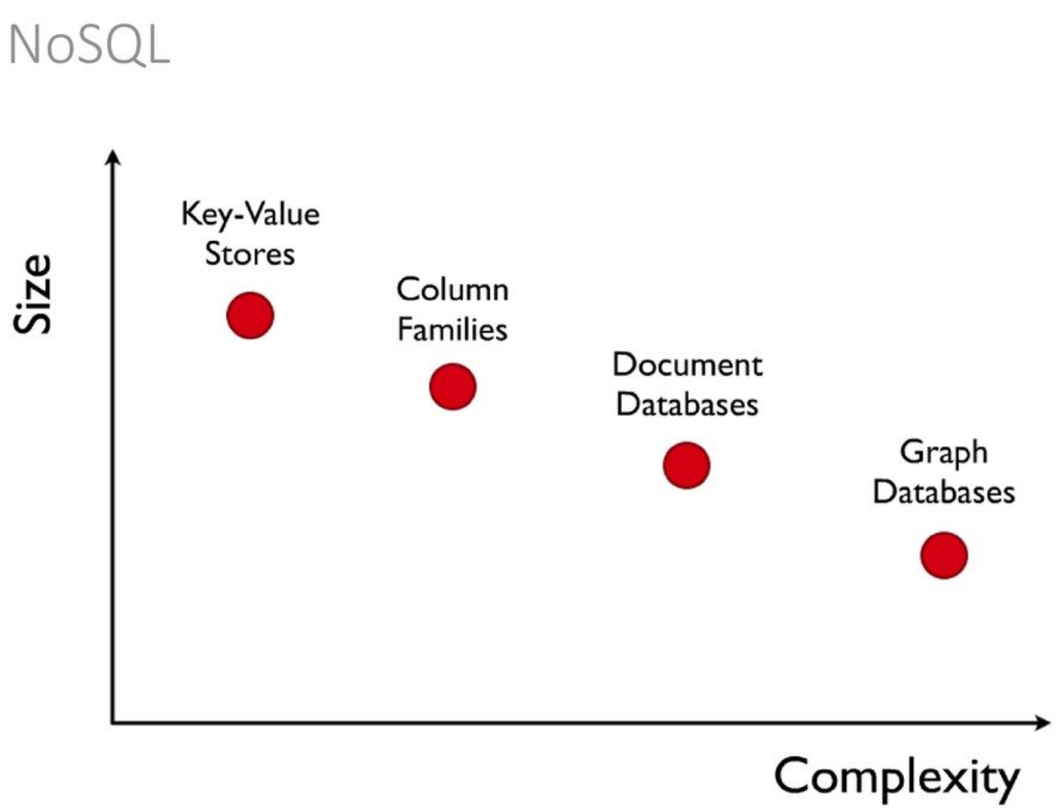
- **Conceptual Level** remains:
  - ER, UML diagram can still be used for no SQL as they output a model that encompasses the whole company.
- **Phsyical Level** remains: NoSQL solutions often expose internals for obtaining flexibility, e.g.,
  - Key-value stores API
  - Column stores
  - Log structures
- *Logical level no longer make sense. Schema on read focuses on the query side._

NoSQL Familty

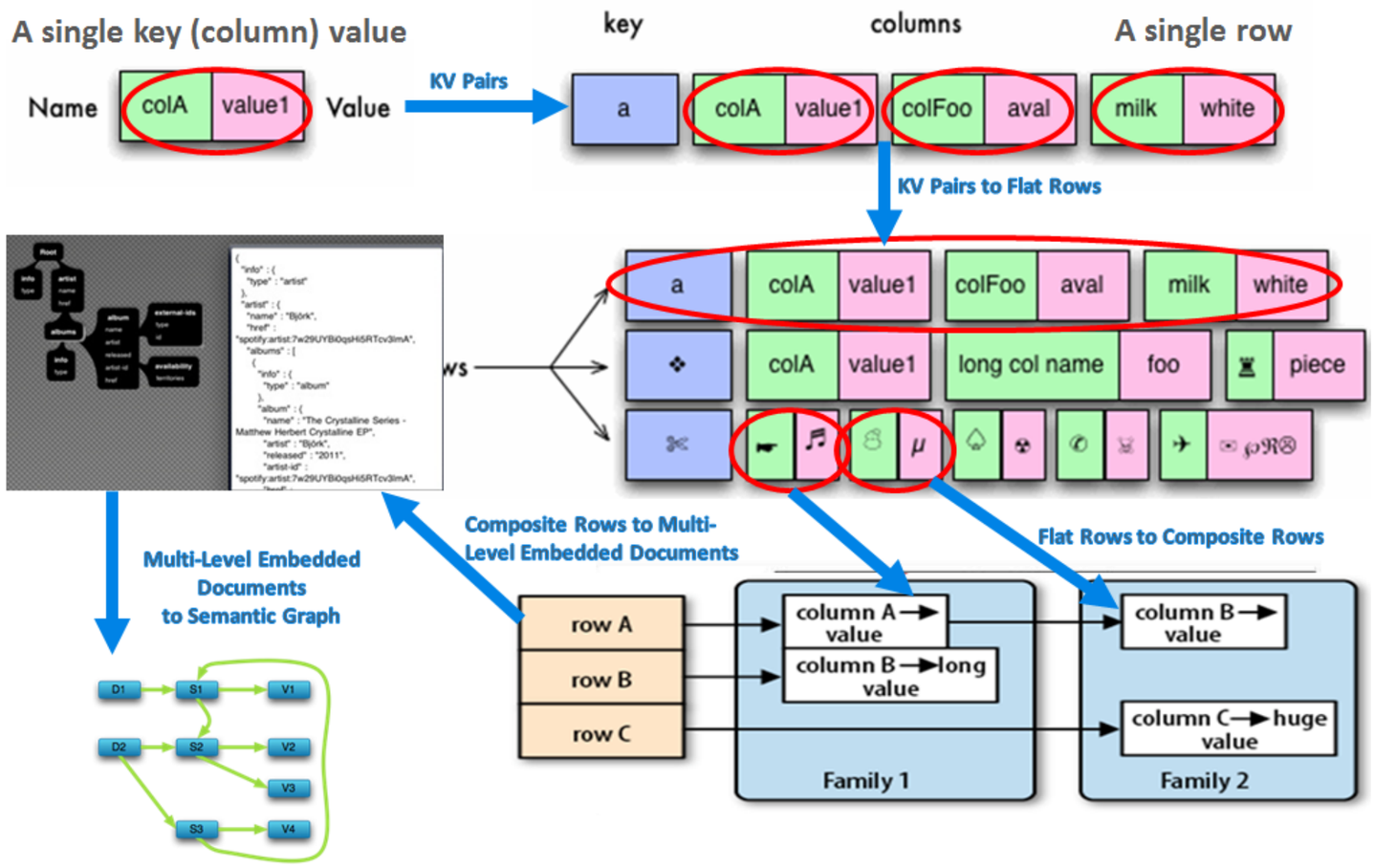
Kinds of NoSQL (2/4)

Kinds of NoSQL (4/4)

Complexity Across Families



Dependencies Across Families

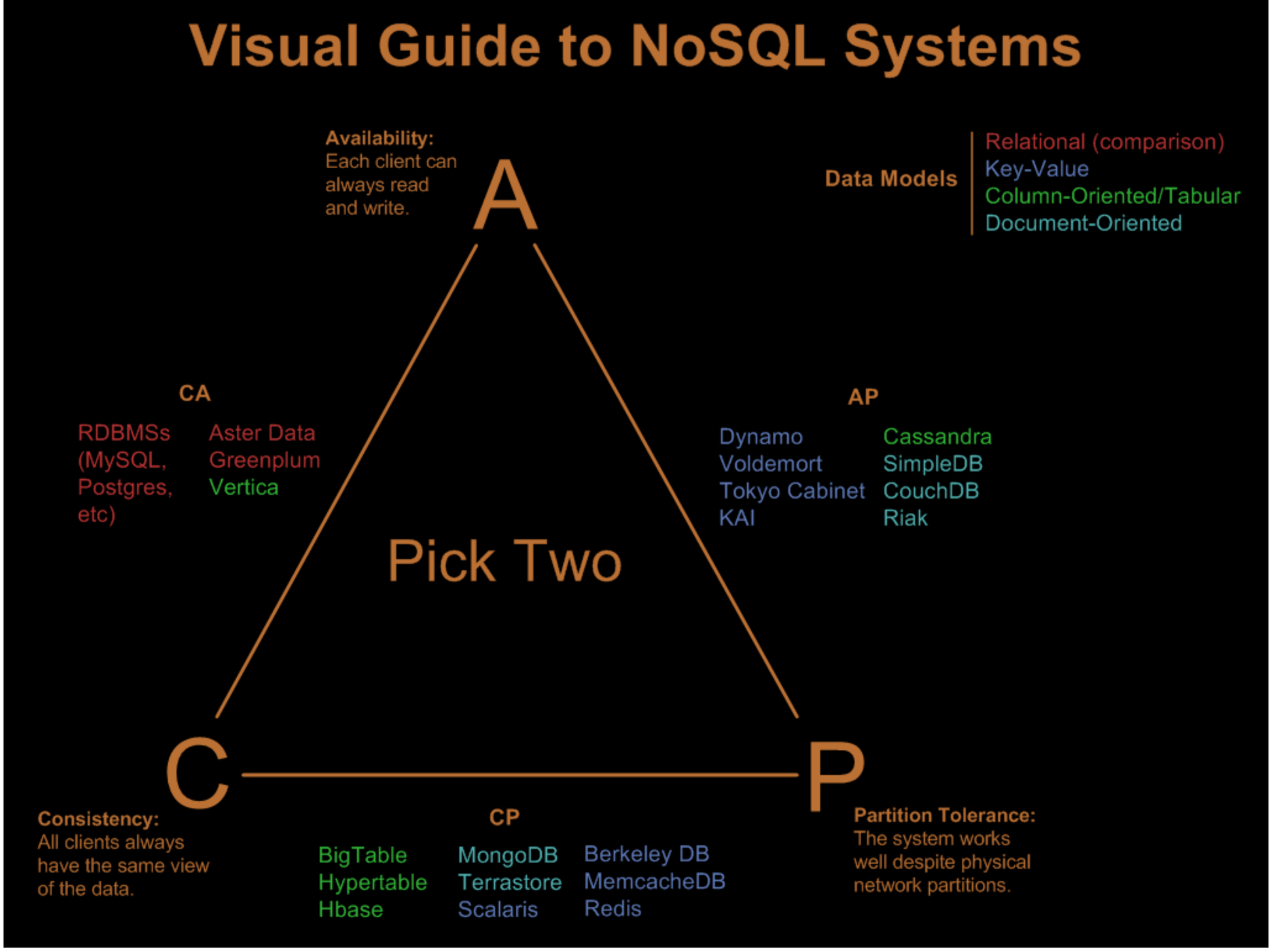


a natural evolutionary path exists from simple key-value stores to the highly complicated graph databases, as shown in the following diagram:

SQL vs NoSQL

SQL databases	NoSQL databases
Triggered the need of relational databases	Triggered by the storage needs of Web 2.0 companies such as Facebook,Google and Amazon.com
Well structured data	Not necessarily well structured – e.g., pictures, documents, web page description, video clips, etc.
Focus on data integrity	focuses on availability of data even in the presence of multiple failures
Mostly Centralised	spread data across many storage systems with a high degree of replication.
ACID properties should hold	ACID properties may not hold[^62]

NoSQL & CAP Theorem


Visual Guide to NoSQL Systems
Availability:
Each client can always read and write.
A
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
C
P
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

img

The ~~OLD~~ ACID Model

- ACID, which stands for Atomicity, Consistency, Isolation, and Durability1-1(app://obsidian.md/index.html#fn-1-799ed3e7c985b657)
- **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.
- ^ The terms was coined in 1983 by Theo Härder and Andreas Reuter 6(app://obsidian.md/index.html#fn-6-799ed3e7c985b657)

Rationale to Change

- It’s ok to use stale data (Accounting systems do this all the time. It’s called “closing out the books.”) ;
- It’s ok to give approximate answers
- Use resource versioning -> say what the data really is about – no more, no less
  - the value of x is 5 at time T

The New BASE Model

BASE(Basically Available, Soft-State, Eventually Consistent)

- **Basic Availability**: fulfill request, even in partial consistency.
- **Soft State**: abandon the consistency requirements of the ACID model pretty much completely
- **Eventual Consistency**: delayed consistency, as opposed to immediate consistency of the ACID properties[67].
  - purely aliveness guarantee (reads eventually return the requested value); but
  - does not make safety guarantees, i.e.,
  - an eventually consistent system can return any value before it converges

---

[67] at some point in the future, data will converge to a consistent state;

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.

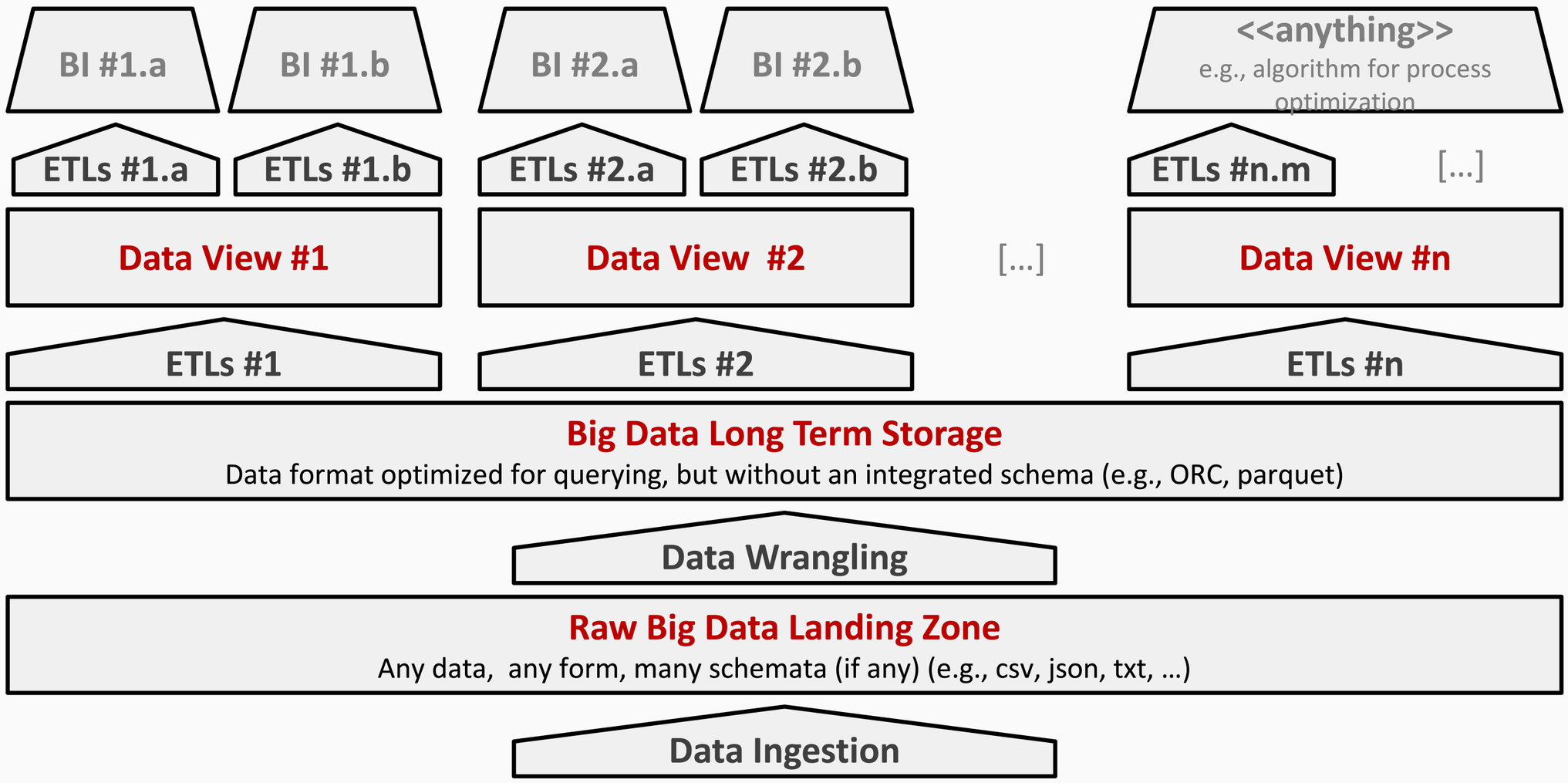
A fully **ACID** database is the perfect fit for use cases where data **reliability** and **consistency** are essential.

Extra Reads

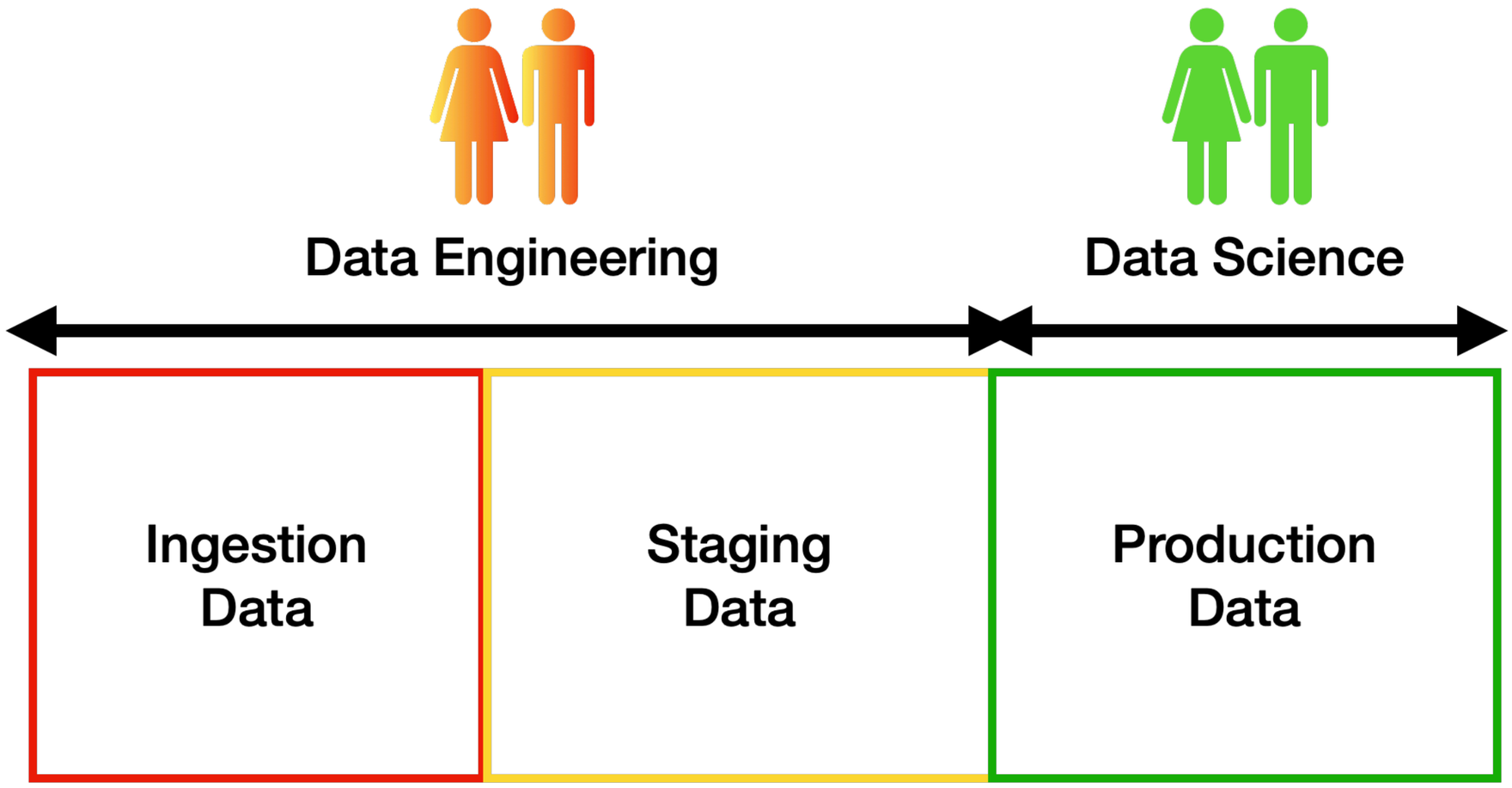
- History of Data Models by Ilya Katsov
- Life beyond Distributed Transactions

Refining the Initial View

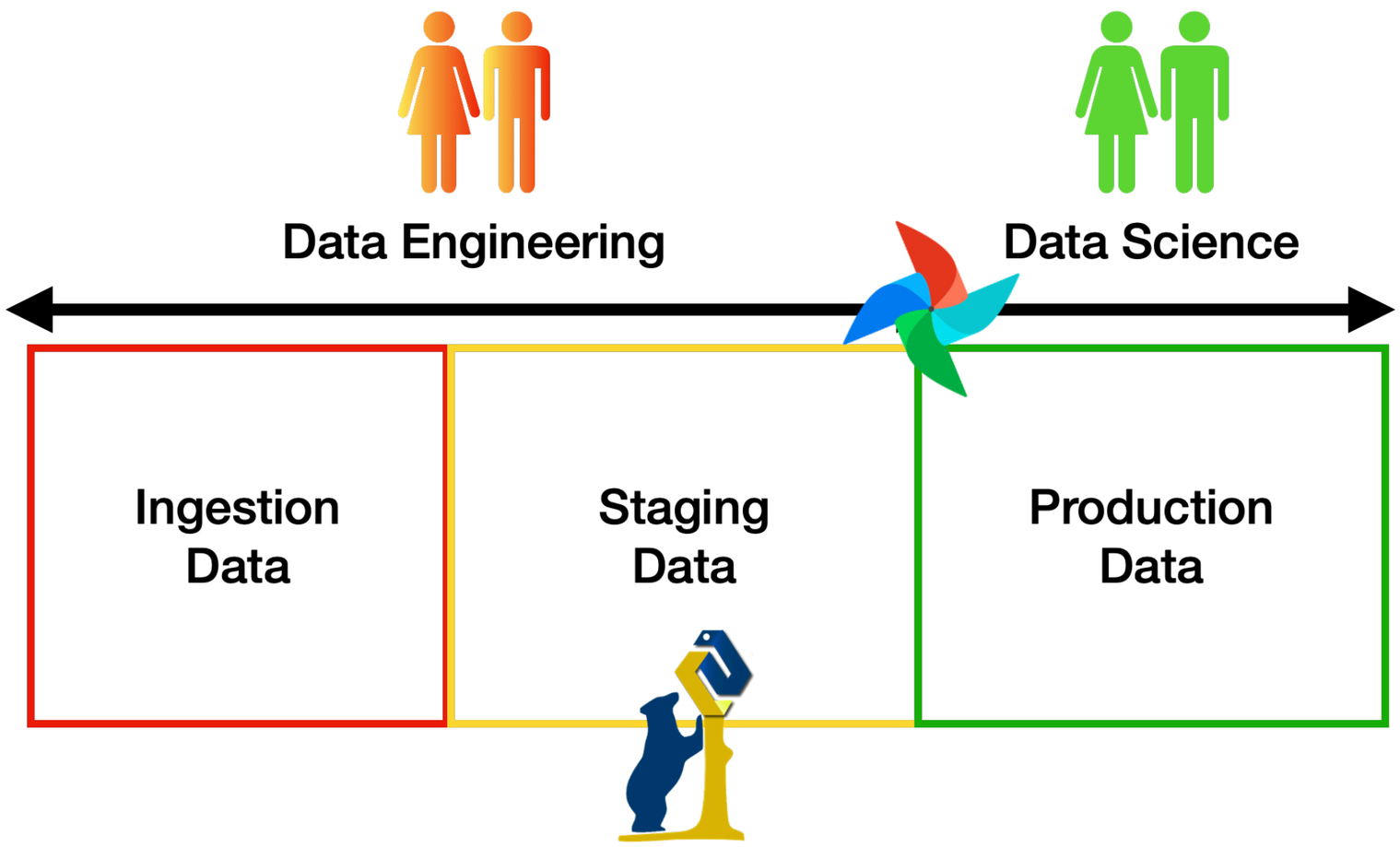
Logical architecture of a data engineering pipeline



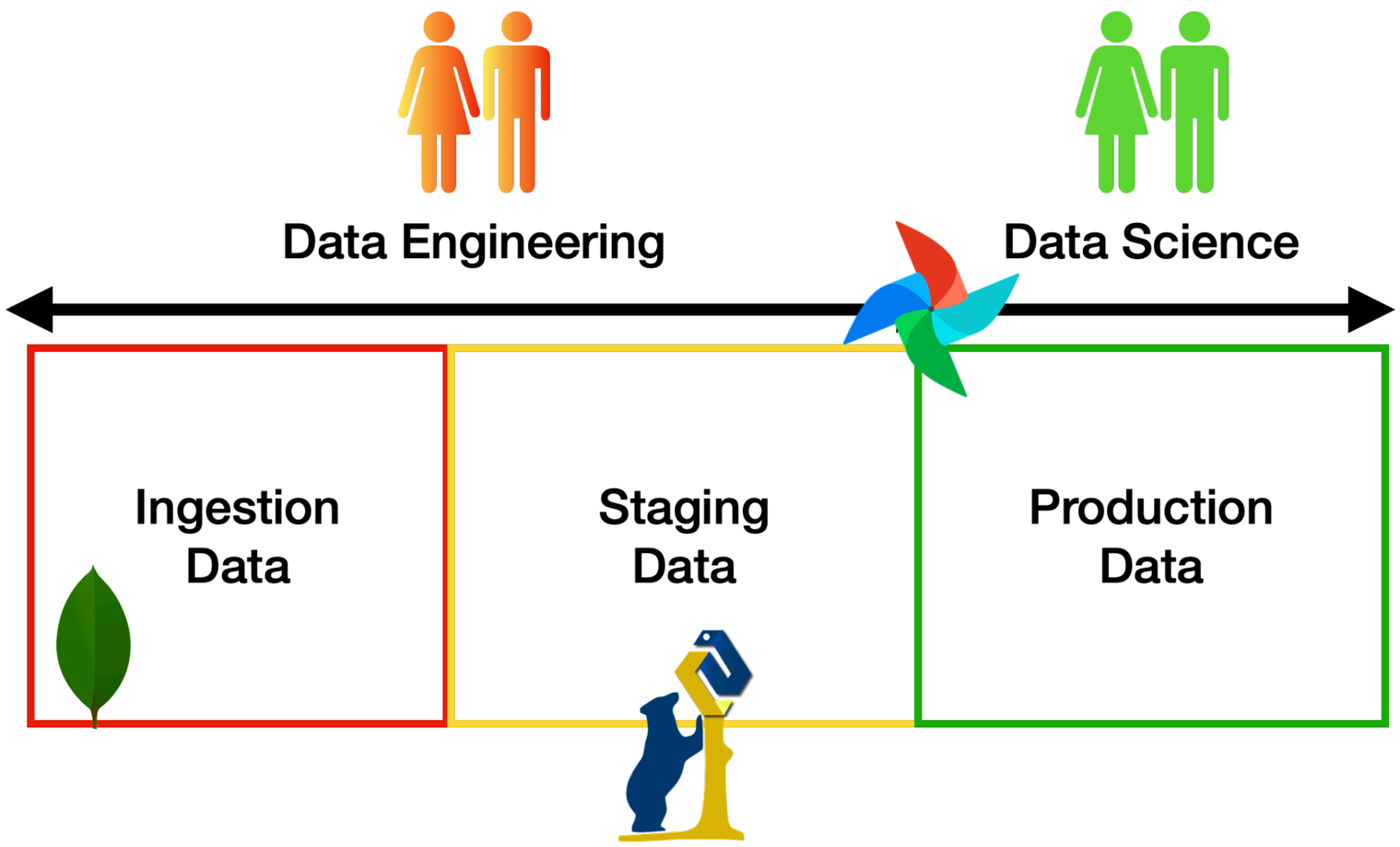
A Simplified view



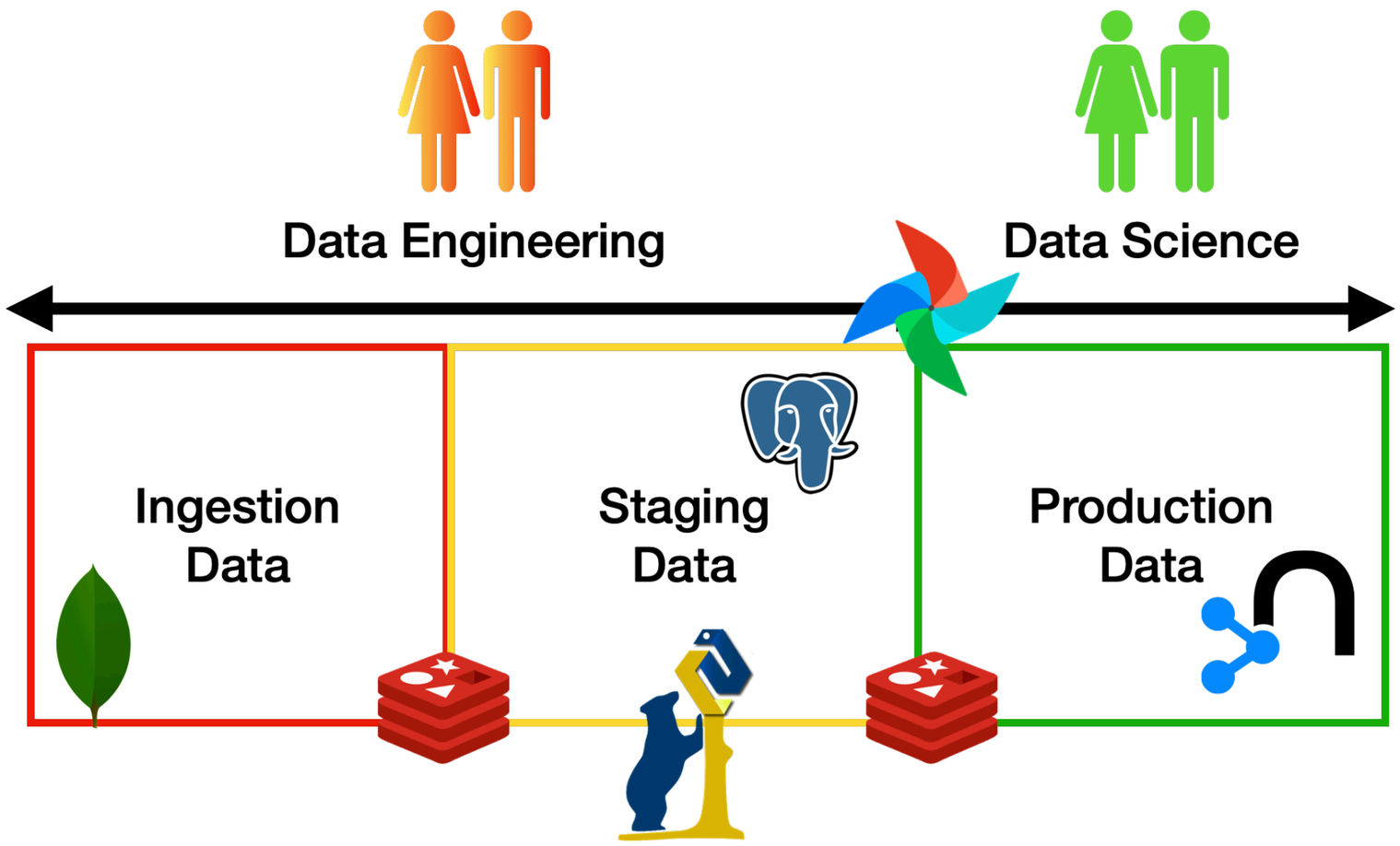
Our Physical View



Our Physical View



Our Physical View



Our Physical View