Data Engineering

LTAT.02.007

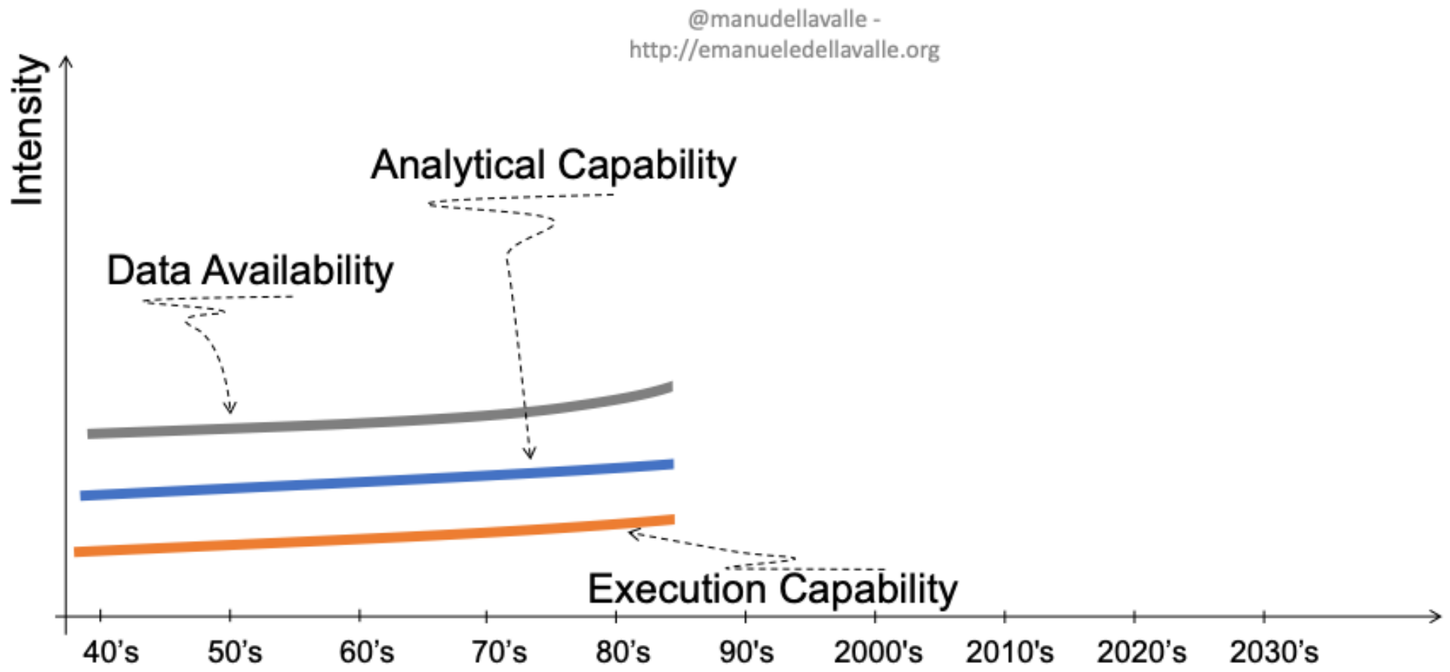
Ass Prof. Riccardo Tommasini

Assistants: Fabiano Spiga, Mohamed Ragab, Hassan Eldeeb

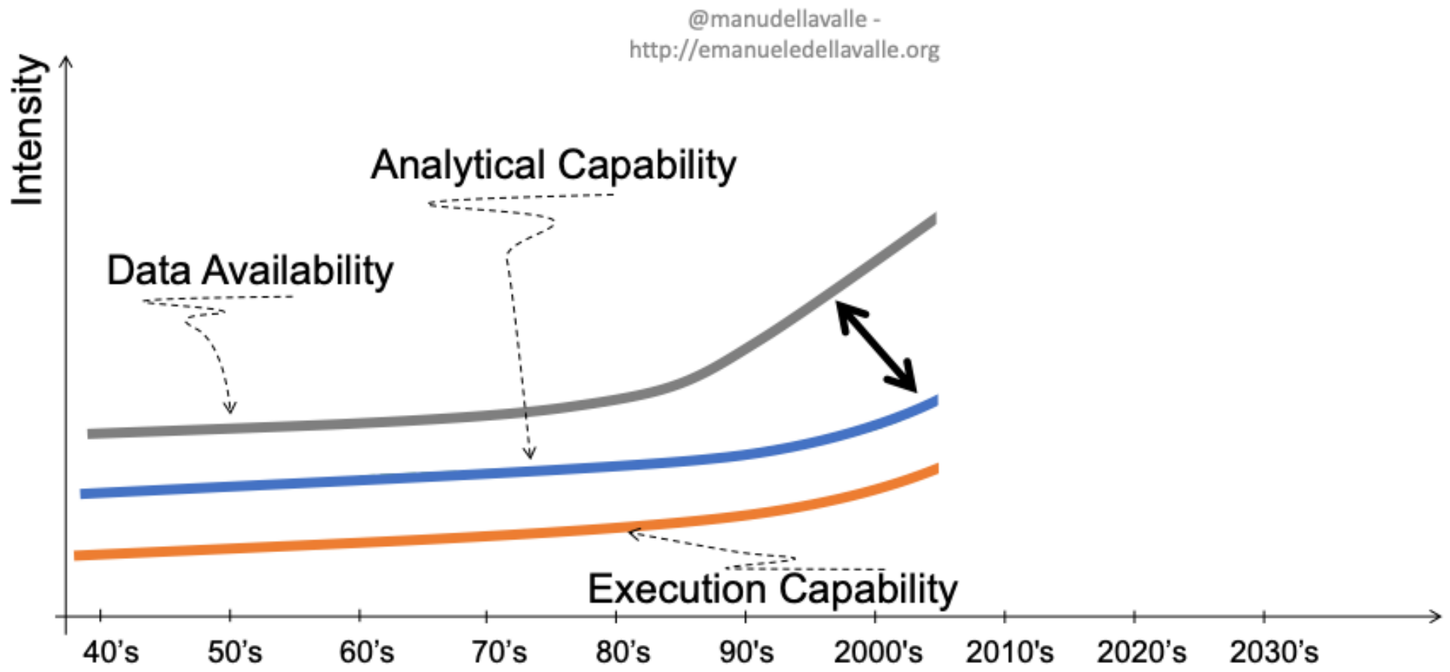
- https://courses.cs.ut.ee/2020/dataeng
- Forum
- Moodle

Data Modeling for Big Data

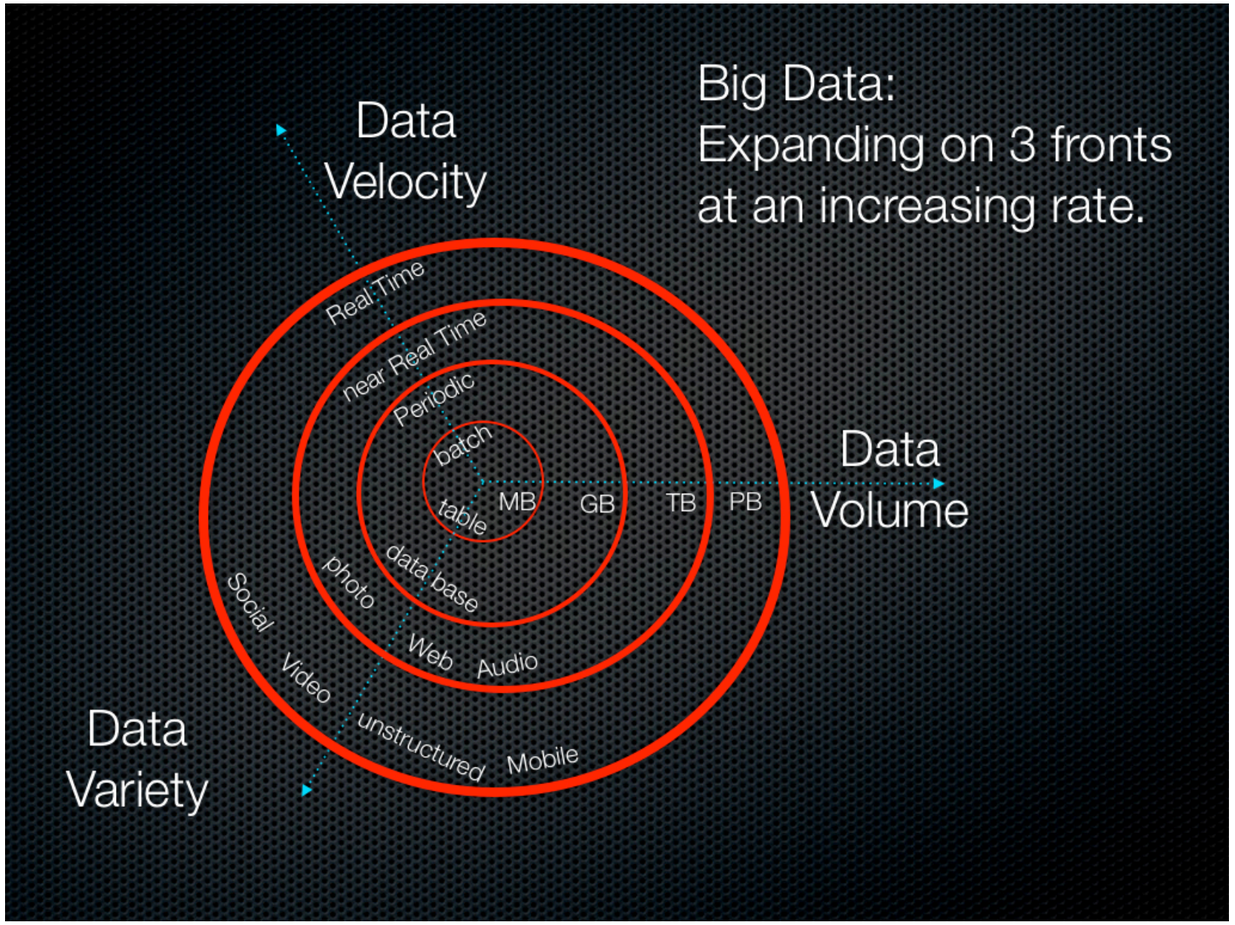
From data to analysis and execution



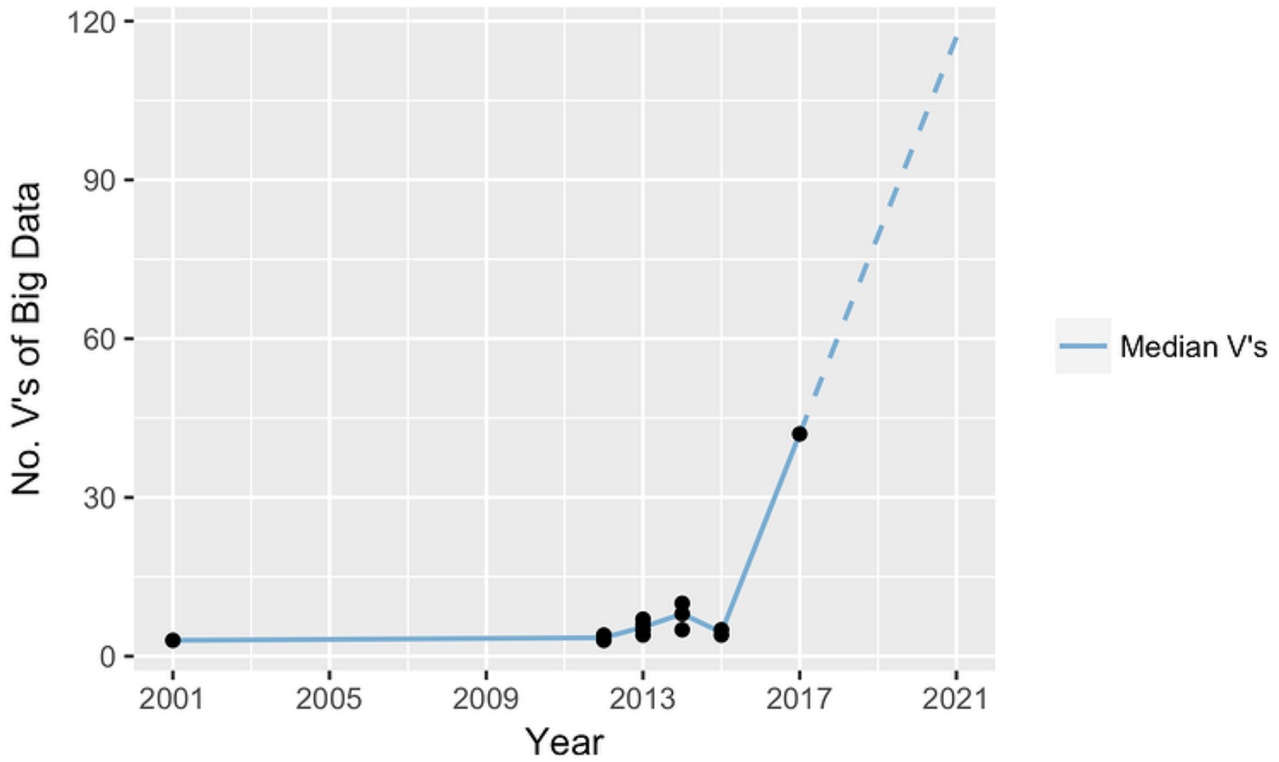
The appearance of the “Big Data”



Big Data Vs [Lanely]

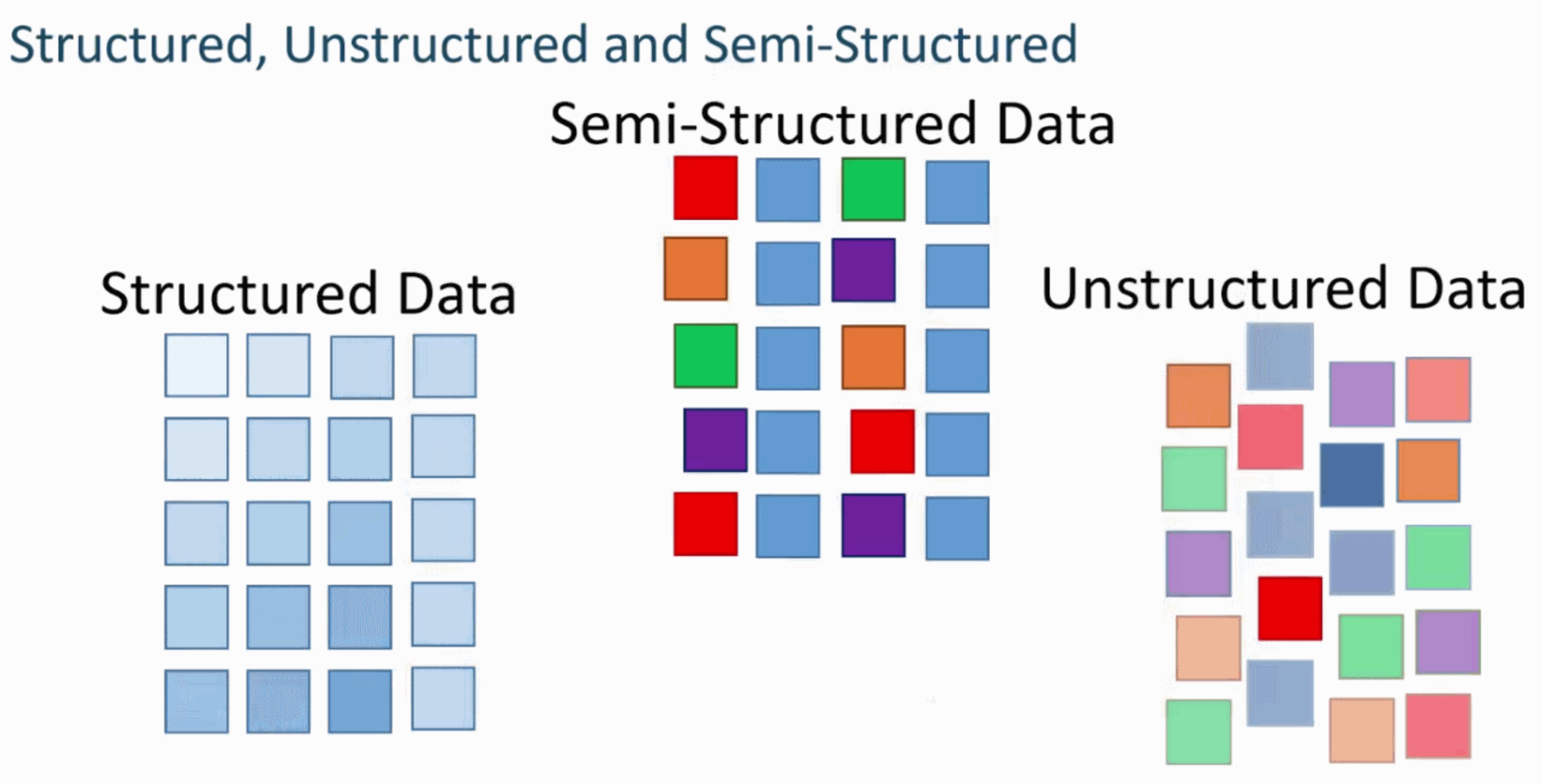


A Growing Trend



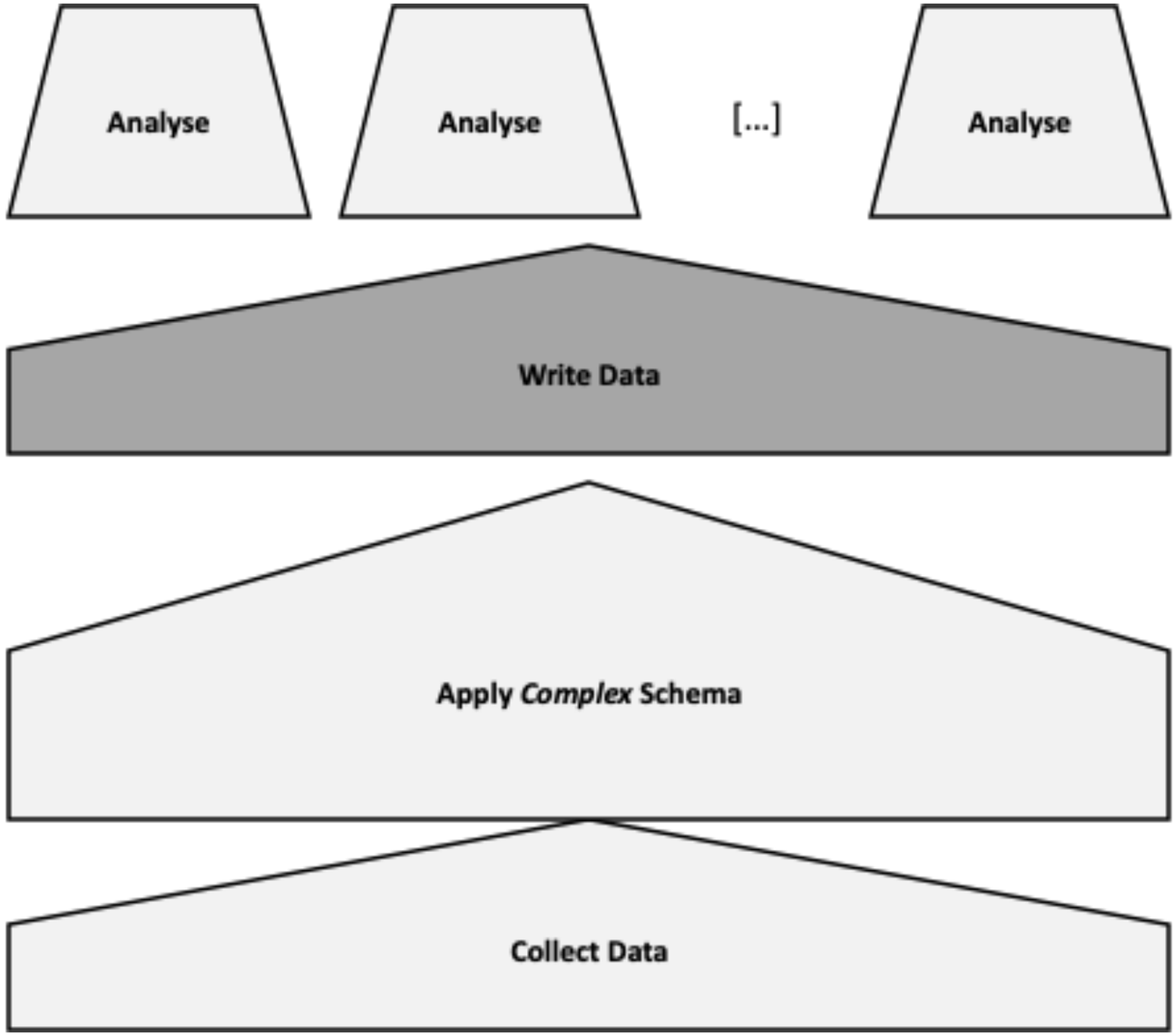
source

The Data Landscape



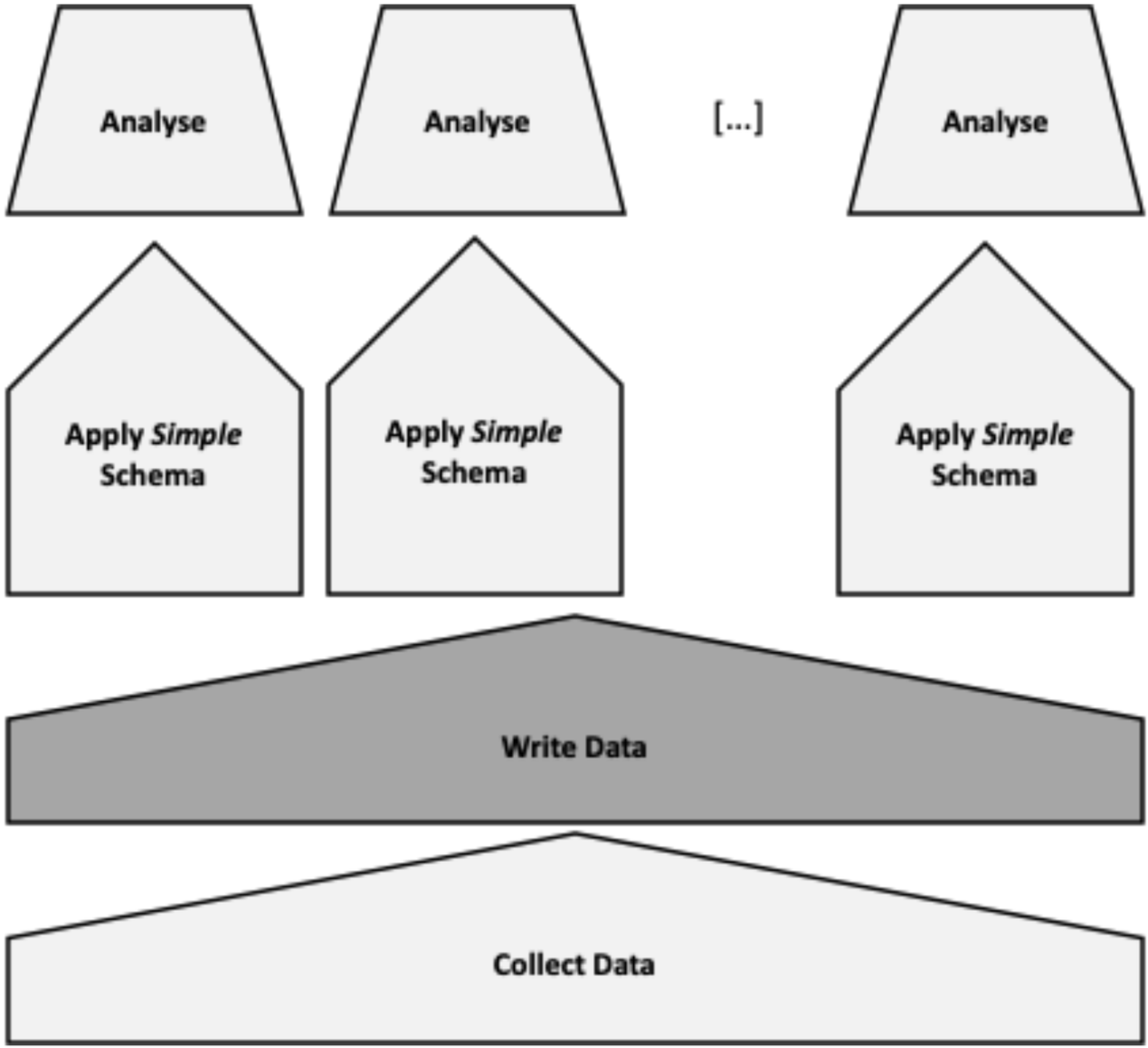
Traditional Data Modelling Workflow

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

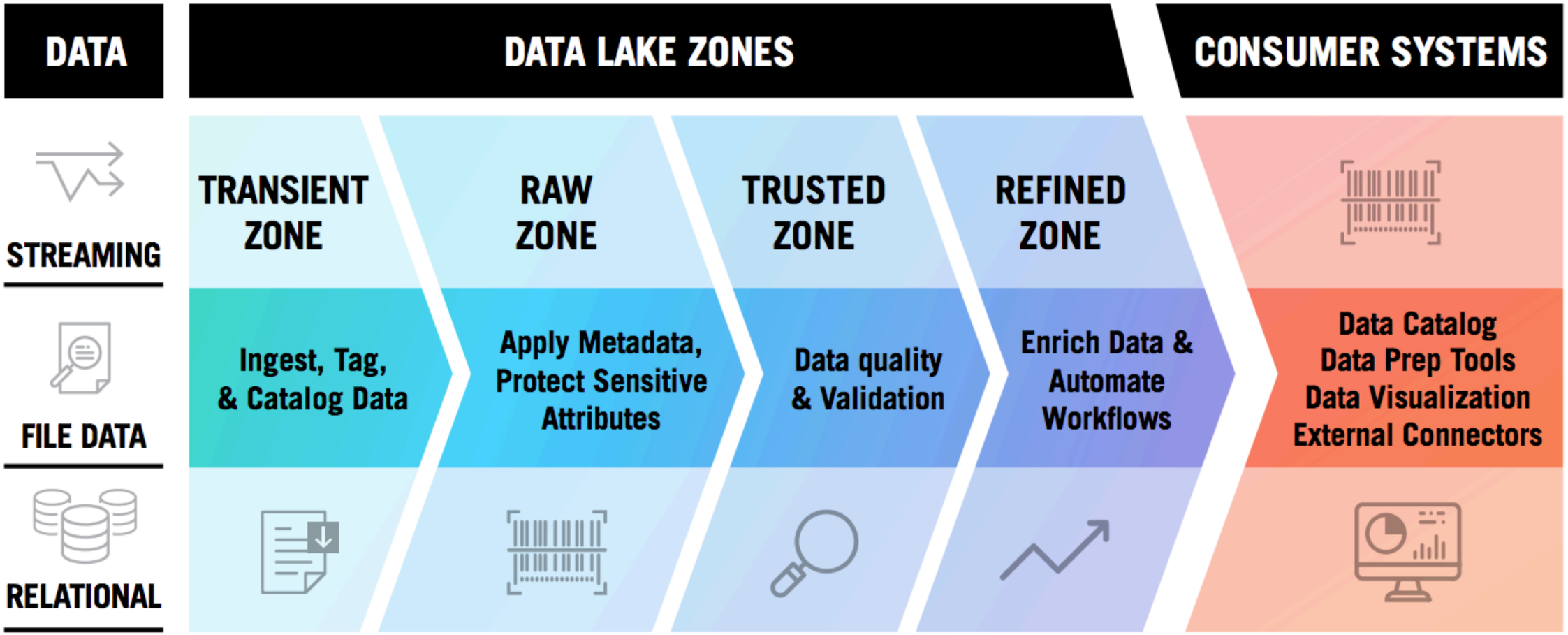


Schema on Read

- 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



Data Lakes


DATA
DATA LAKE ZONES
CONSUMER SYSTEMS
STREAMING
FILE DATA
RELATIONAL
TRANSIENT ZONE
RAW ZONE
TRUSTED ZONE
REFINED ZONE
Ingest, Tag, & Catalog Data
Apply Metadata, Protect Sensitive Attributes
Data quality & Validation
Enrich Data & Automate Workflows
Data Catalog
Data Prep Tools
Data Visualization
External Connectors

Horizontal vs Vertical Scalability

Introduction

- "Traditional" SQL system scale **vertically** (scale up) - Adding data to a "traditional" SQL system may degrade its performances
  - When the machine, where the SQL system runs, no longer performs as required, the solution is to buy a better machine (with more RAM, more cores and more disk)
- Big Data solutions scale **horizontally** (scale out)
  - Adding data to a Big Data solution may degrade its performances
  - When the machines, where the big data solution runs, no longer performs as required, the solution is to add another machine

hardware

Commodity

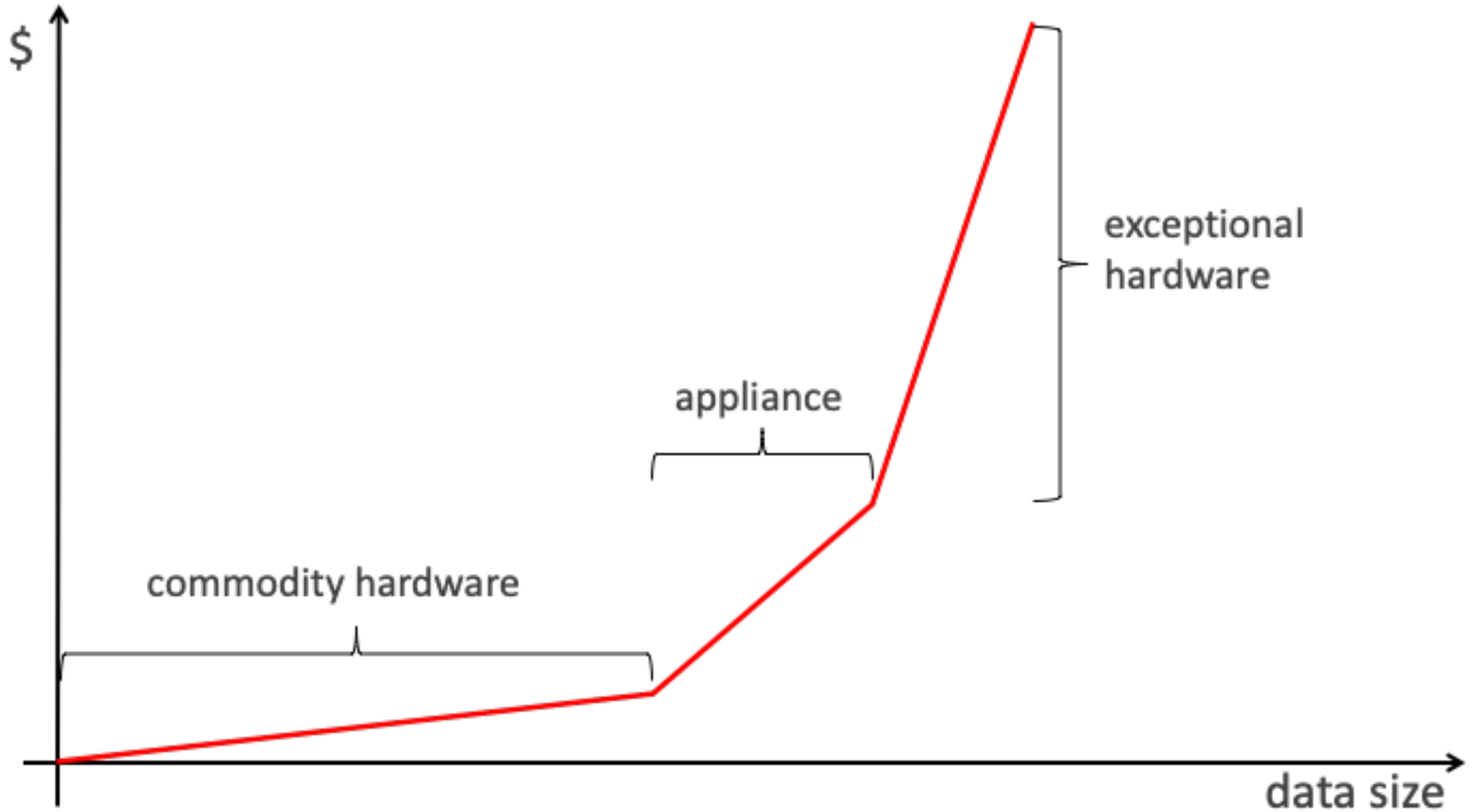
- CPU: 8-32 cores
- RAM: 16-64 GB
- Disk: 1-3 TB
- Network: 10 GE

Appliance

- CPU: 576 cores
- RAM: 24TB
- Disk: 360TB of SSD/rack
- Network: 40 Gb/second InfiniBand

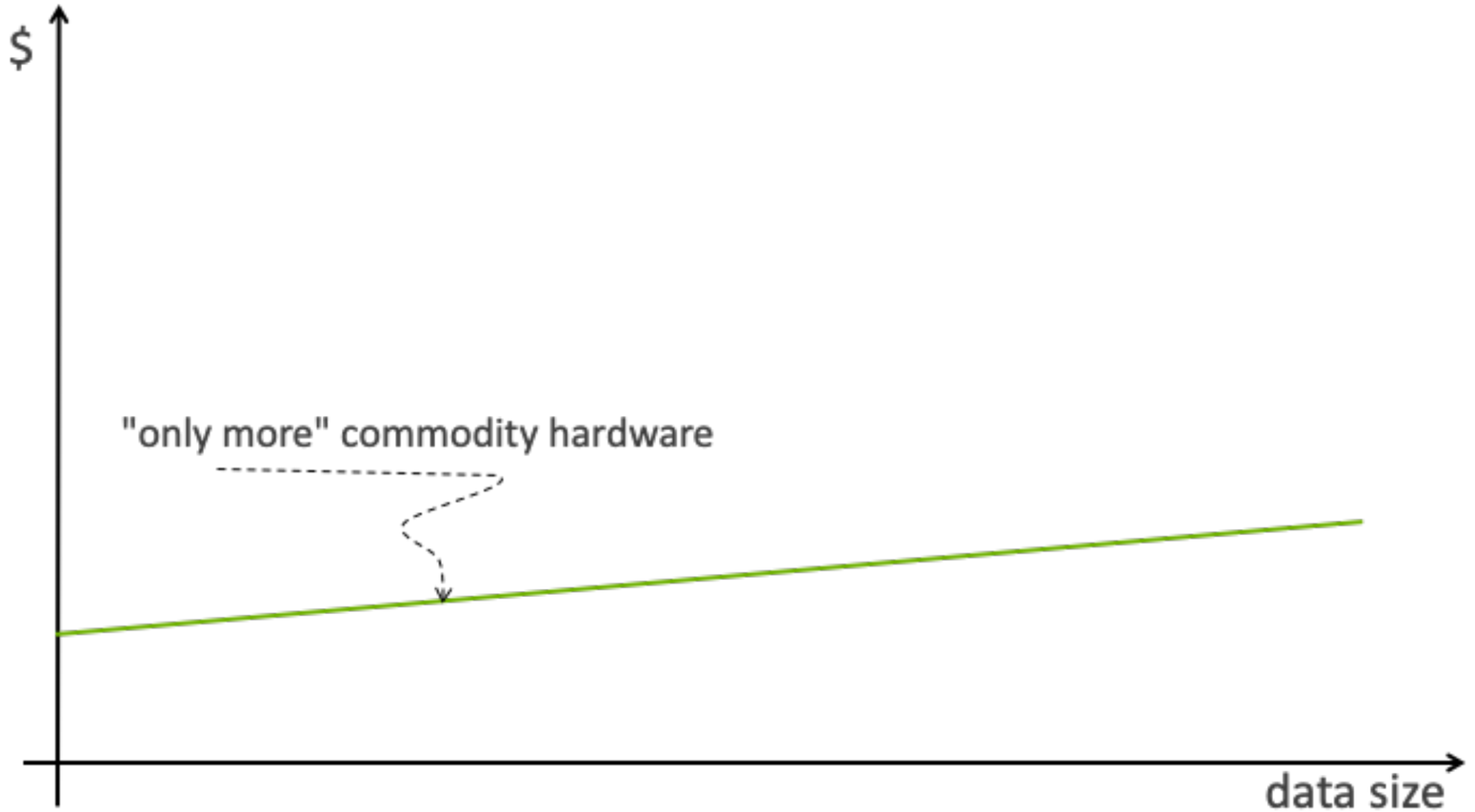
Curtesy of Emanuele Della Valle and Marco Brambilla

Vertical Scalability



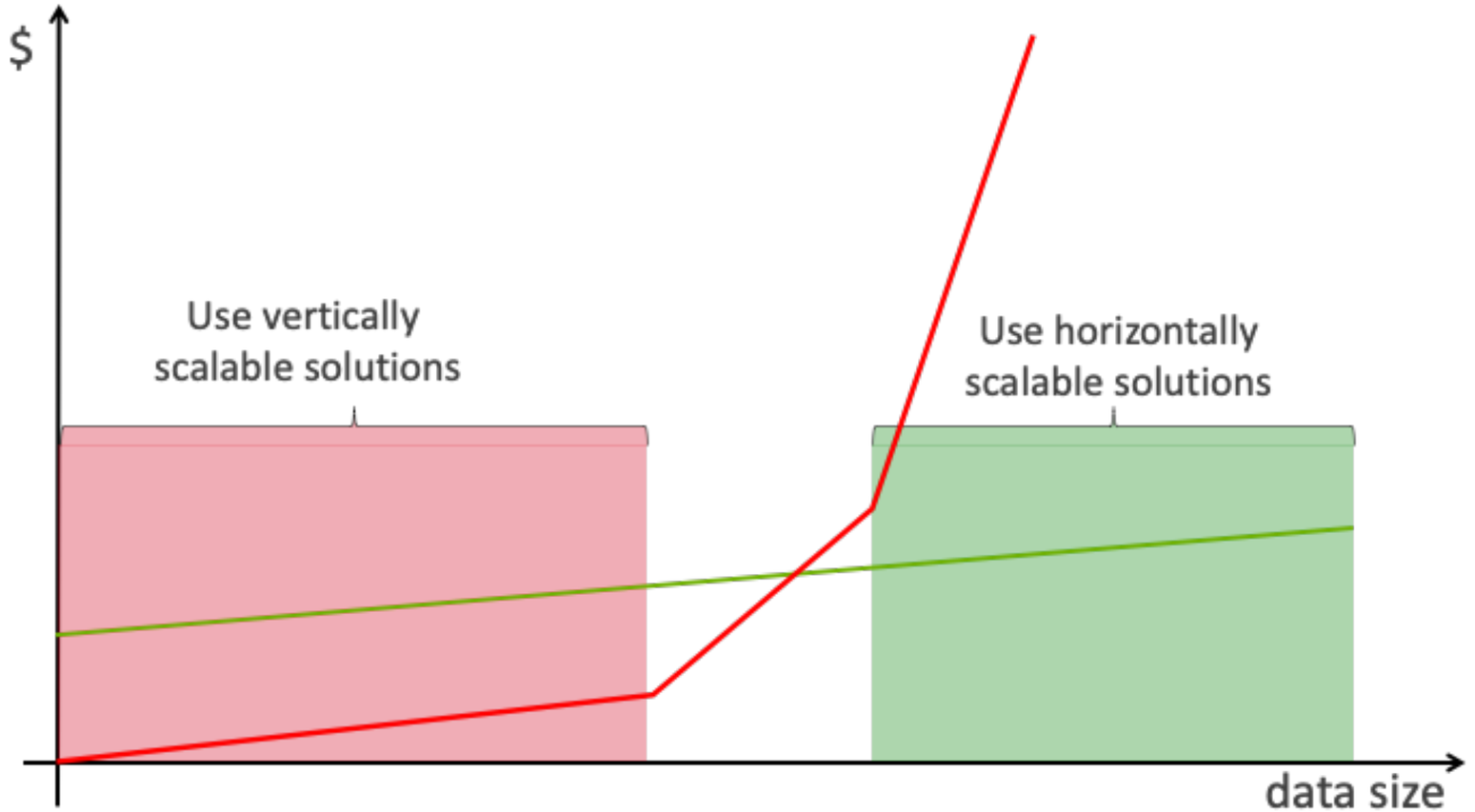
Curtesy of Emanuele Della Valle and Marco Brambilla

Horizontal Scalability



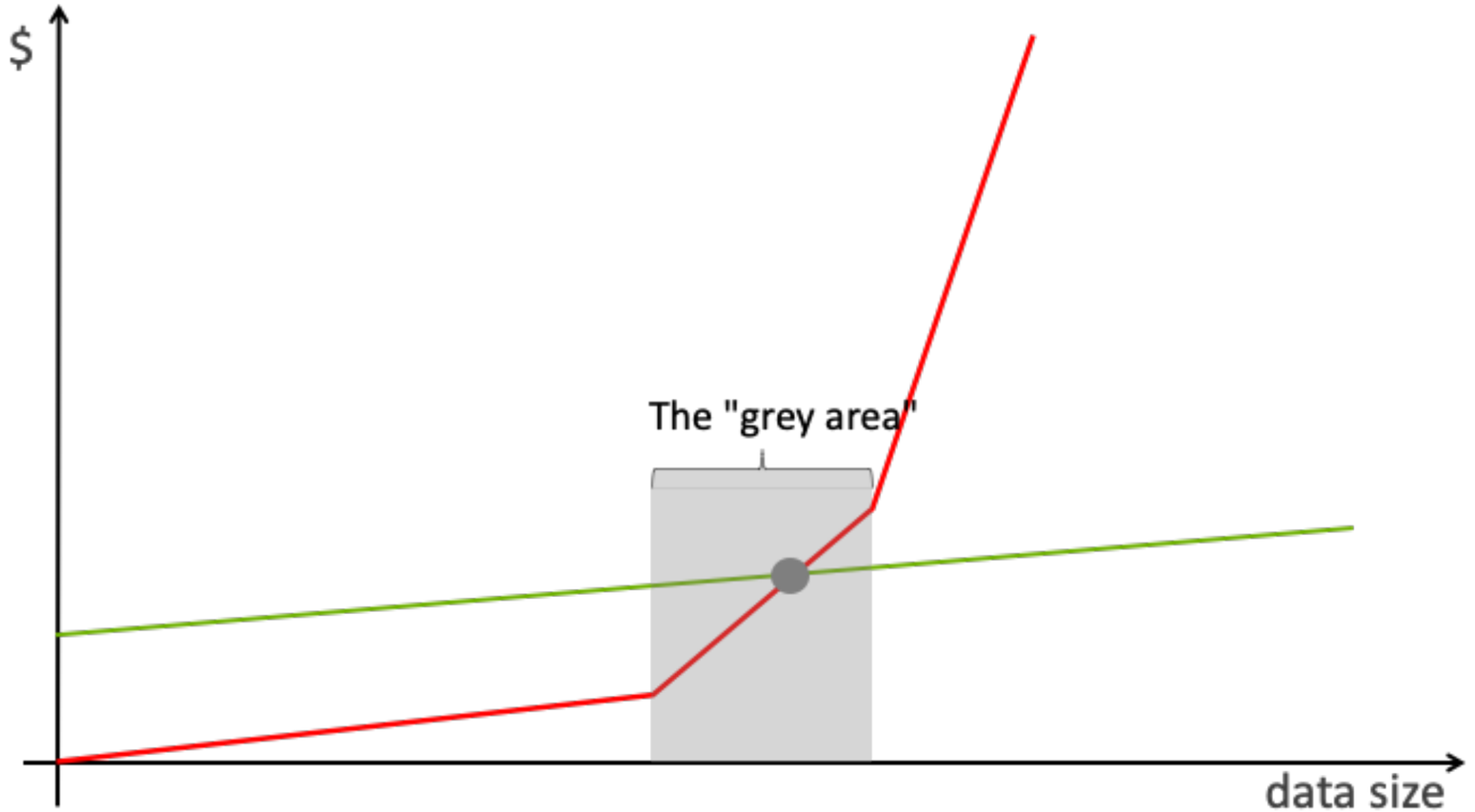
Curtesy of Emanuele Della Valle and Marco Brambilla

Vertical vs Horizontal Scalability



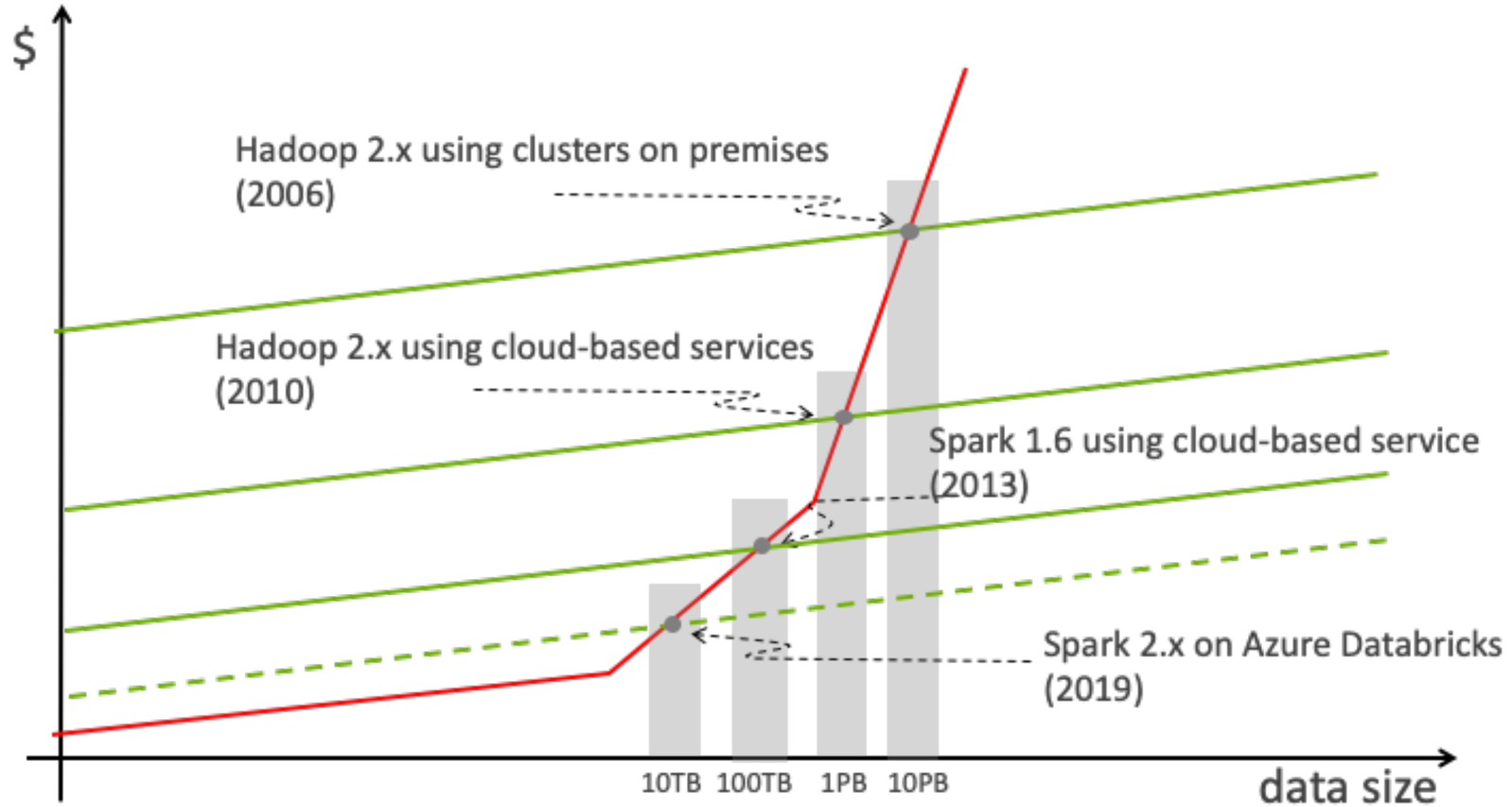
Curtesy of Emanuele Della Valle and Marco Brambilla

Vertical vs Horizontal Scalability



Curtesy of Emanuele Della Valle and Marco Brambilla

Grey Area is Time-Dependent



Curtesy of Emanuele Della Valle and Marco Brambilla

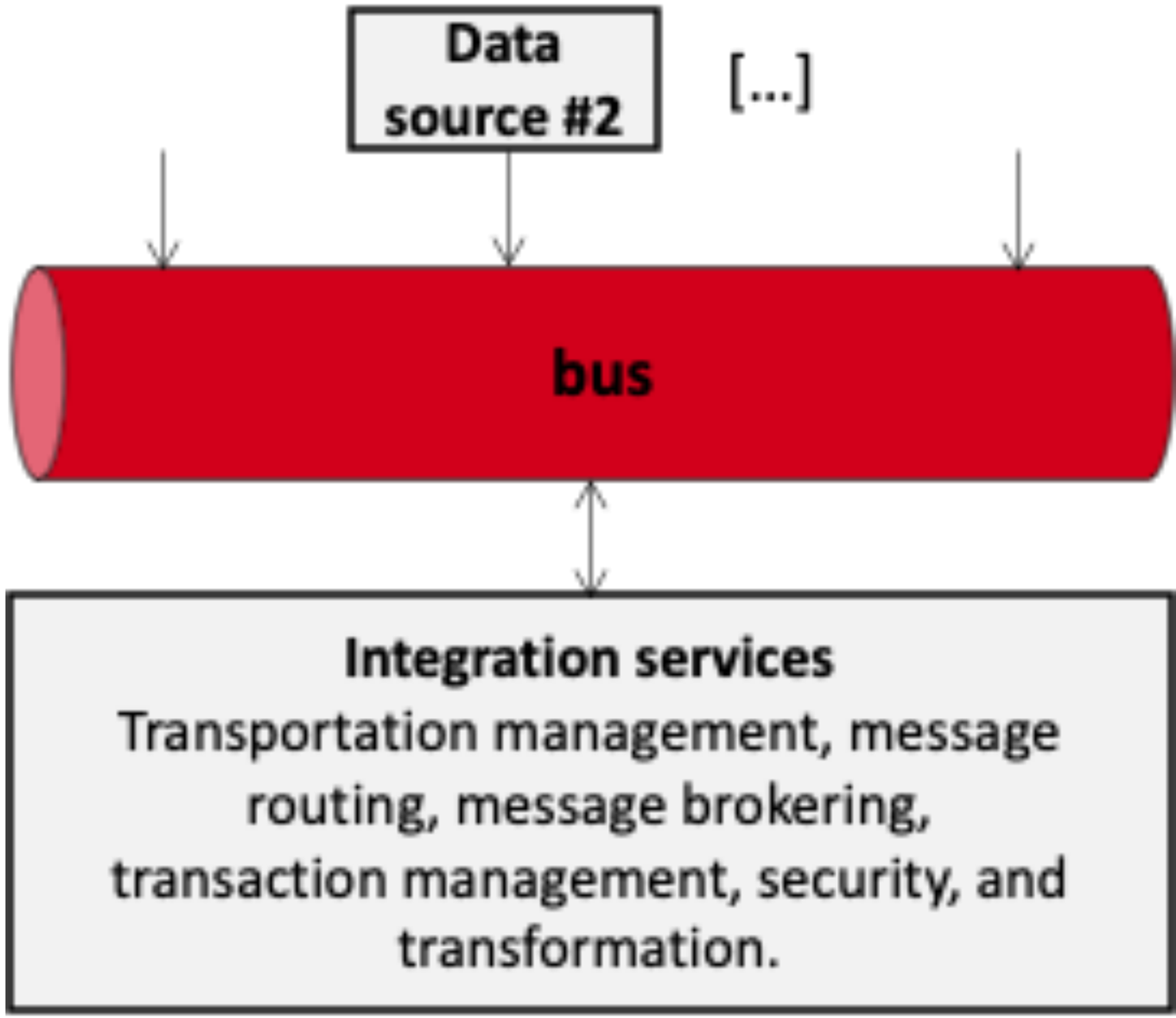
Big Data Storage

- Distributed File Systems, e.g., HDFS
- NoSQL Databases
- NewSQL Databases[65] e.g., VoltDB
- Distributed Queues, e.g., Pulsar or Kafka

[65] a modern form of relational databases that aim for comparable scalability with NoSQL databases while maintaining the transactional guarantees made by traditional database systems

Data Ingestion

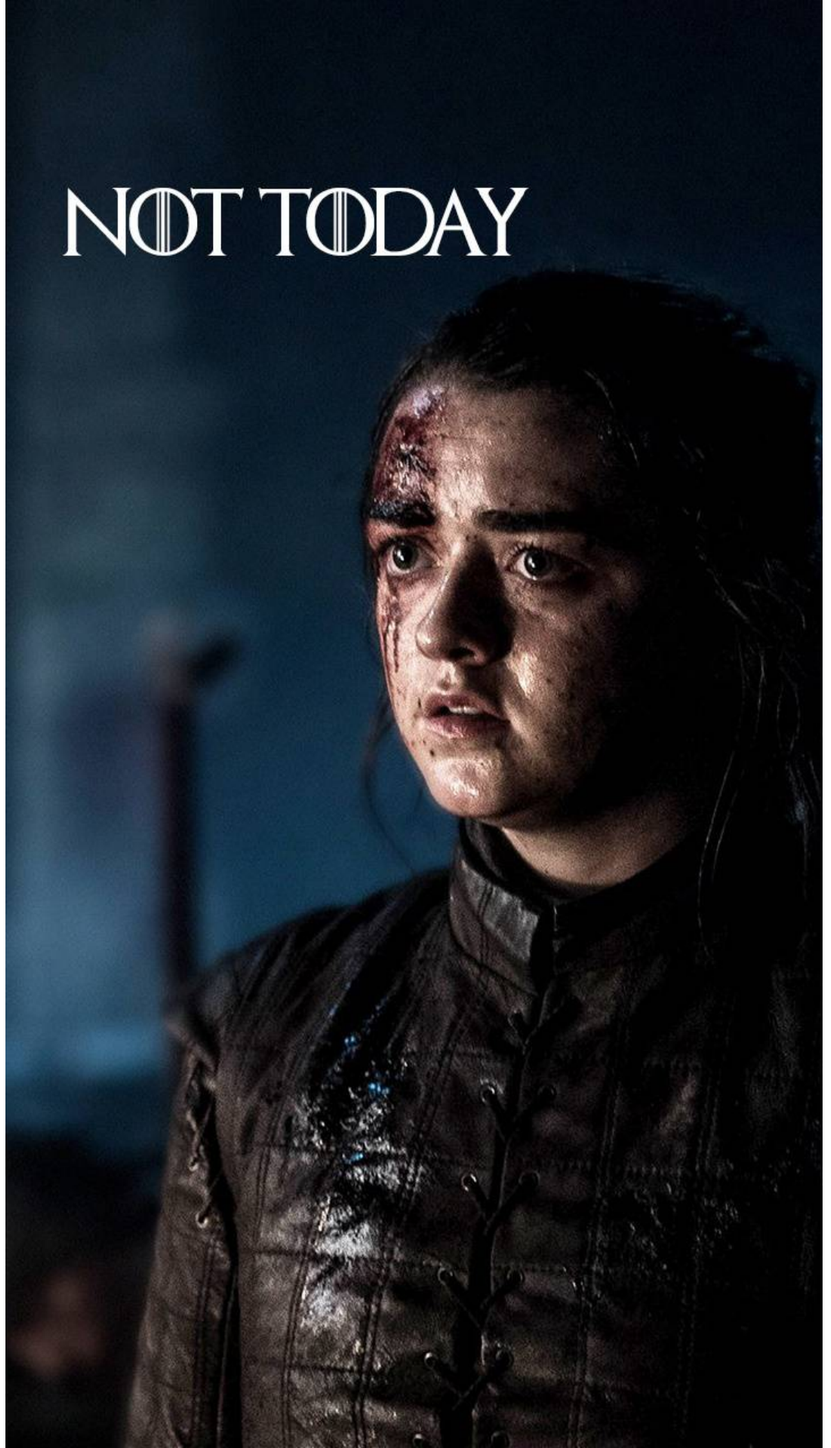
- The process of importing, transferring and loading data for storage and later use
- It involves loading data from a variety of sources
- It can involve altering and modification of individual files to fit into a format that optimizes the storage
- For instance, in Big Data small files are concatenated to form files of 100s of MBs and large files are broken down in files of 100s of MB



We Will Talk About Distributed File Systems

A distributed file system stores files across a large collection of machines while giving a single-file-system view to clients.

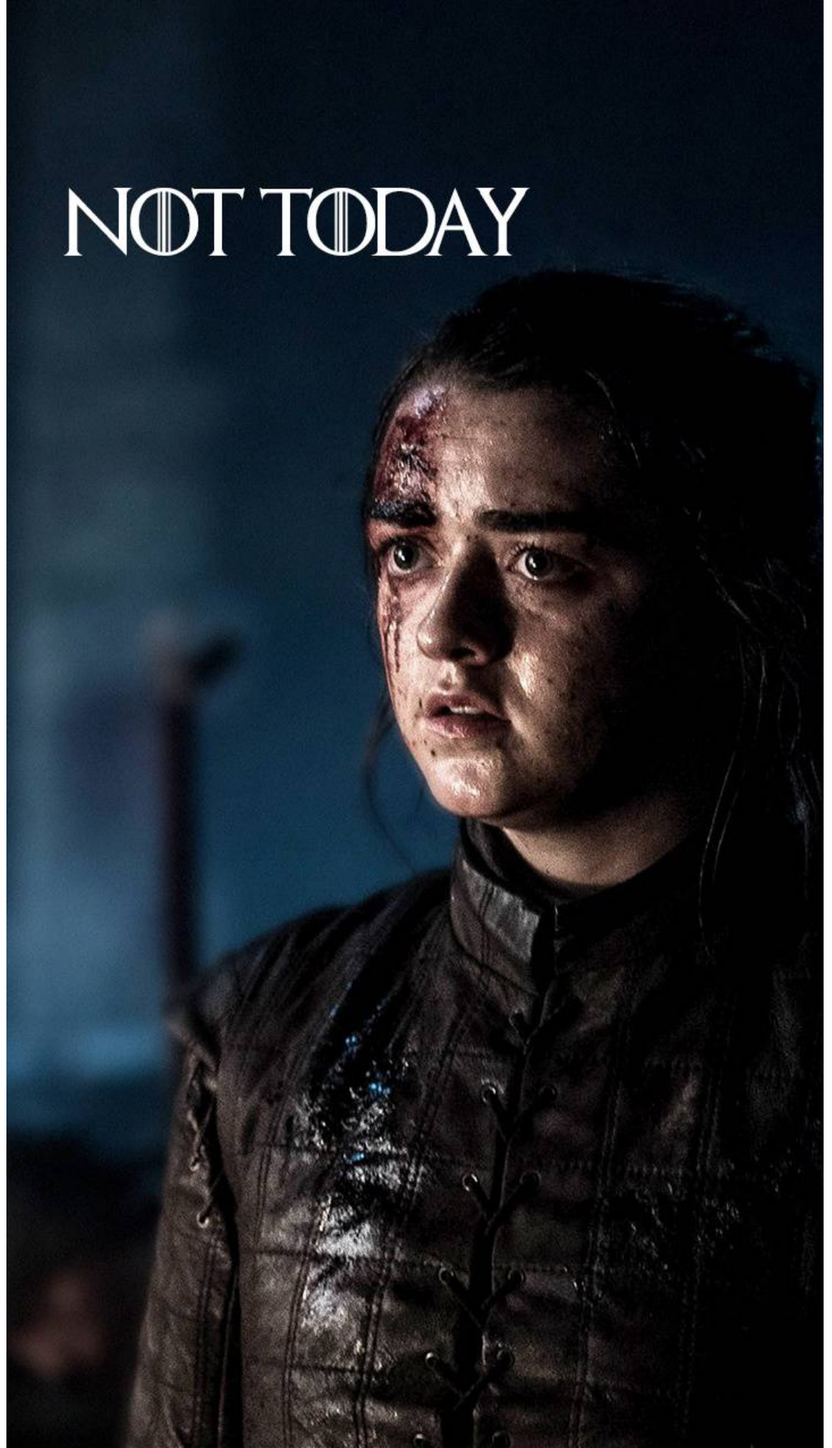
- ![[HDFS]]



Will Will Talk About Distributed Message Queues

A distribured message quque stores file in a log an allos sequential reads.

- ![[Apache Kafka]]



~~ETL~~ [[Data Pipelines]]

A data pipeline aggregates, organizes, and moves data to a destination for storage, insights, and analysis.

Modern data pipeline generalize the notion of ETL (extract, transform, load) to include data ingestion, integration, and movement across any cloud architecture and add additional layers of resiliency against failure.

- [[Apache Airflow]]
- [[Kafka Streams]]
- [[KSQL]]

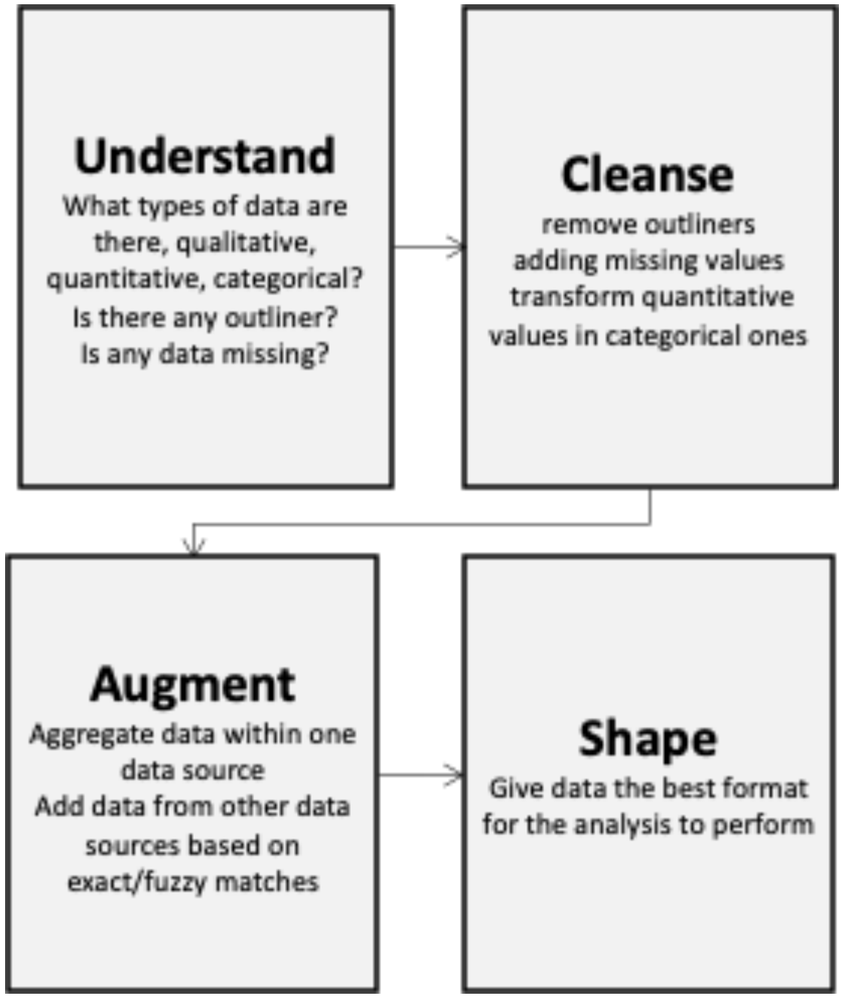
[[Data Wrangling]]

The process of creating *reliable* that can be analysed to generate valid actionable insights.

The central goal is to make data usable: to put data in a form that can be easily manipulated by analysis tools.

It includes understanding, cleansing, augmenting and shaping data.

The results is data in the best format (e.g., columnar) for the analysis to perform.



The Advent of NoSQL

Quote time

Google, Amazon, Facebook, and DARPA all recognized 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 Reasons Behind

- **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
- **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

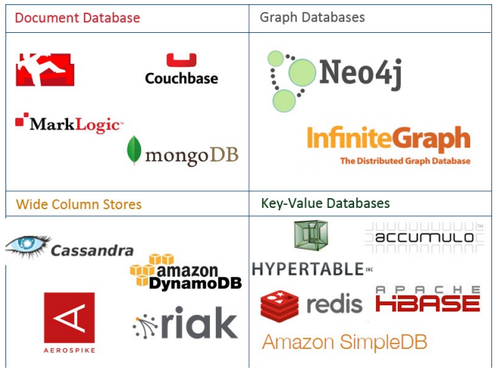
Most application development today is done in **object-oriented** programming languages

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

NoSQL Timeline

NoSQL Familty



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

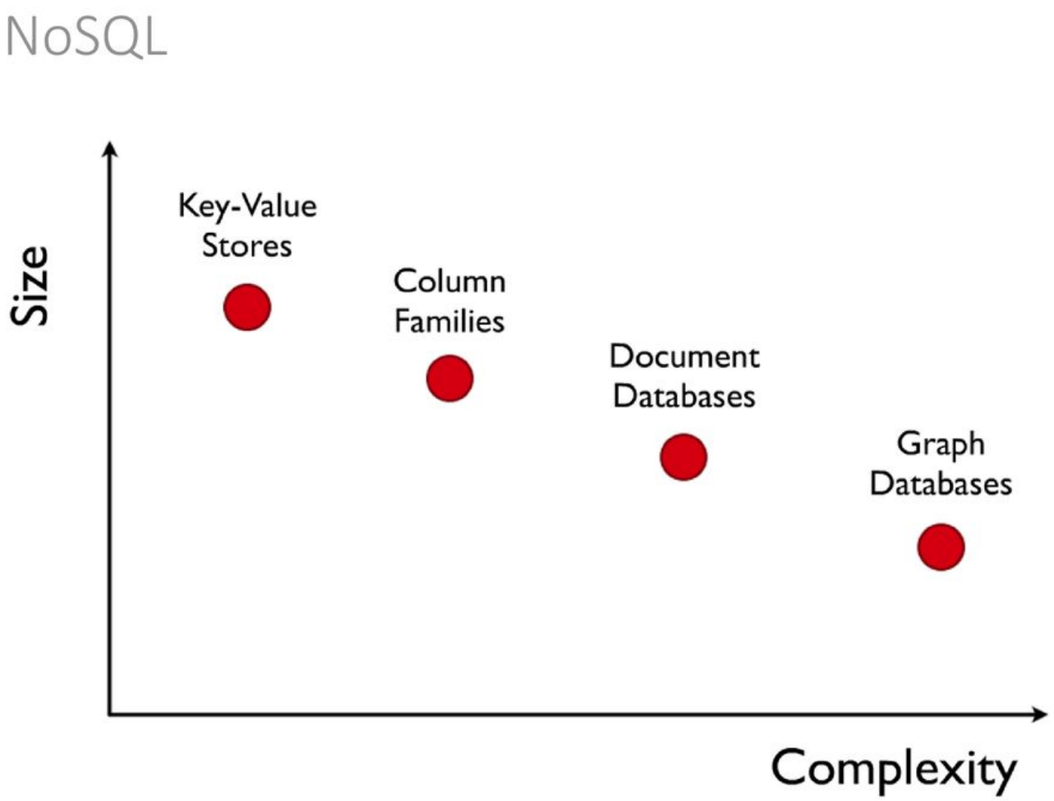
Kinds of NoSQL (4/4)

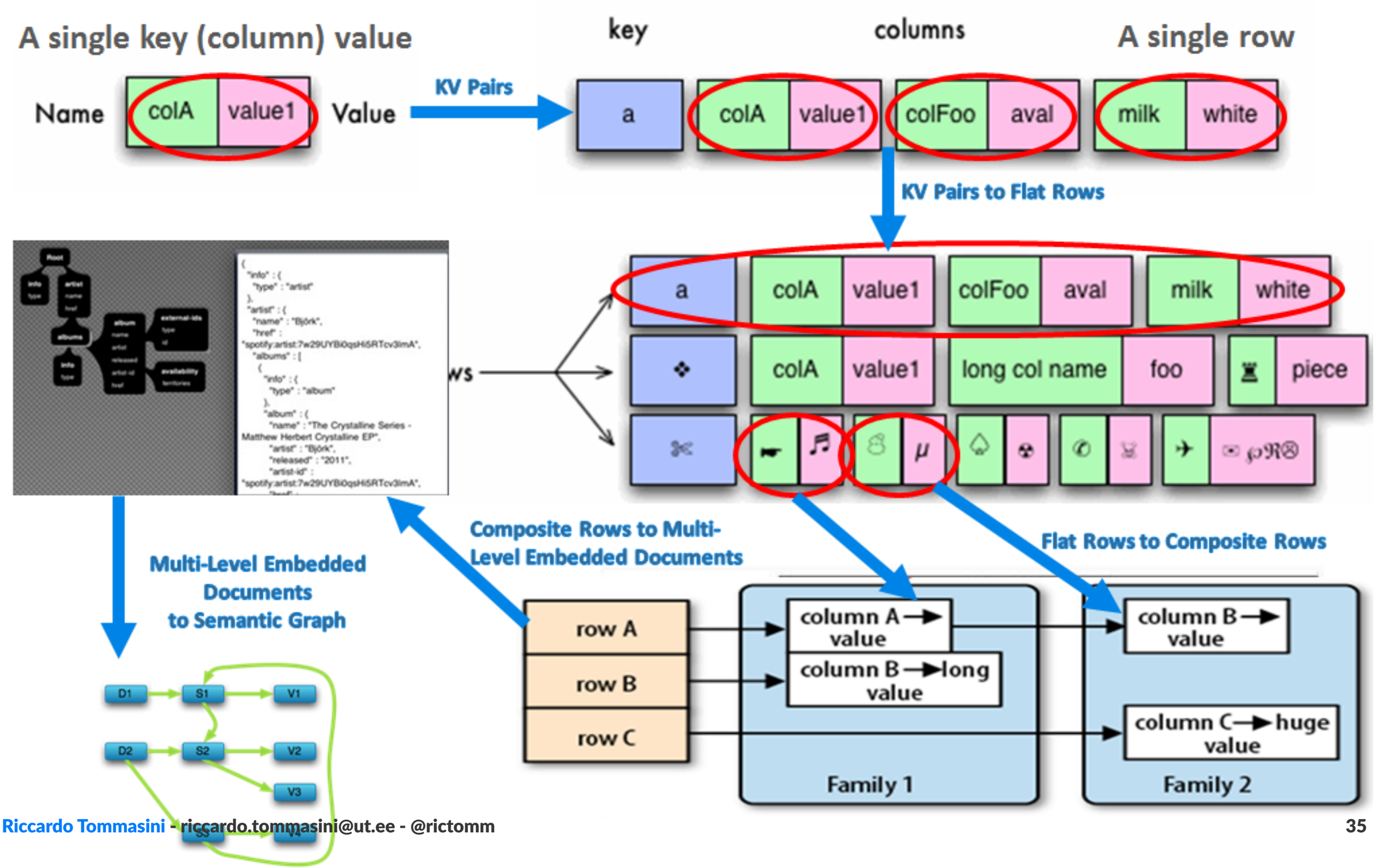
- **Document / XML / Object Store**
  - Key (and possibly other indexes) point at a serialized object
  - DB can operate against values in document
  - Examples: MongoDB, CouchDB, RavenDB
- **Graph Store**
  - Nodes are stored independently, and the relationship between nodes (edges) are stored with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

You can also distinguish them

- **Key/Value or 'the big hash table' (remember caching?)**
  - Amazon S3 (Dynamo)
  - Voldemort
  - Scalaris
  - MemcacheDB,
  - Azure Table Storage,
  - *Redis* ←
  - Riak
- **Schema-less**
  - *MongoDB* ←
  - Cassandra (column-based)
  - CouchDB (document-based)
  - *Neo4J (graph-based)* ←
  - HBase (column-based)

NoSQL Complexity




A single key (column) value
Name
colA
value1
Value
KV Pairs
key
columns
A single row
a
colA
value1
colFoo
aval
milk
white
KV Pairs to Flat Rows
{
"info" : {
"type" : "artist"
},
"artist" : {
"name" : "Björk",
"href" :
"spotify:artist:7w29UYBiOqsHi5RTcv3lmA",
"albums" : [
{
"info" : {
"type" : "album"
},
"album" : {
"name" : "The Crystalline Series -
Matthew Herbert Crystalline EP",
"artist" : "Björk",
"released" : "2011",
"artist-id" :
"spotify:artist:7w29UYBiOqsHi5RTcv3lmA",
a
colA
value1
colFoo
aval
milk
white
colA
value1
long col name
foo
piece
Multi-Level Embedded Documents to Semantic Graph
Composite Rows to Multi-Level Embedded Documents
Flat Rows to Composite Rows
row A
row B
row C
column A value
column B long value
column B value
column C huge value
Family 1
Family 2
D1
S1
V1
D2
S2
V2
V3

SQL vs (Not only SQL) NoSQL

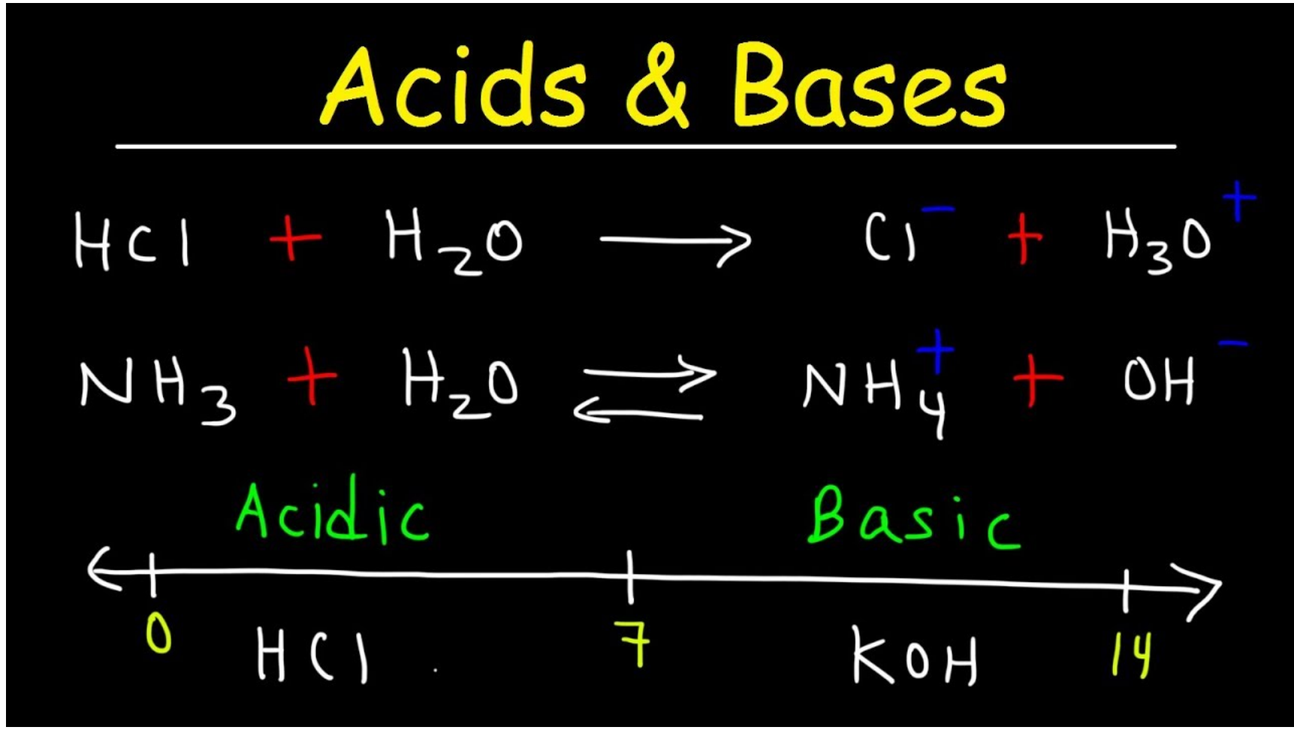
SQL databases	NoSQL databases
Triggered the need of relational databases	Triggered by the storage needs of Web 2.0 companies such as Facebook,Google and Amazon.com
Well structured data	Not necessarily well structured – e.g., pictures, documents, web page description, video clips, etc.
Focus on data integrity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

ACID vs. BASE properties[61]

[61] Do you recall the CAP theorem? 🎩

Rationale

- 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

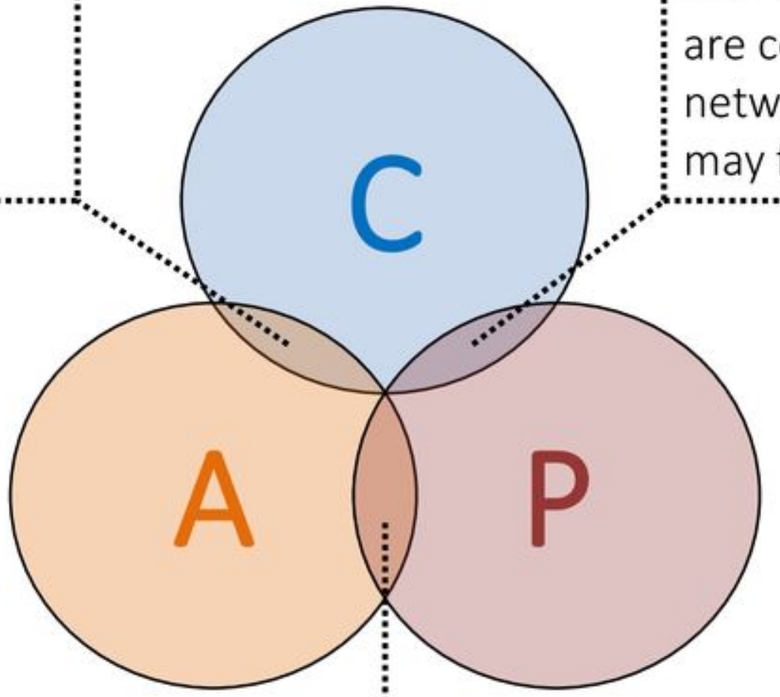


CAP Theorem is a Trade-off, remember?

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)

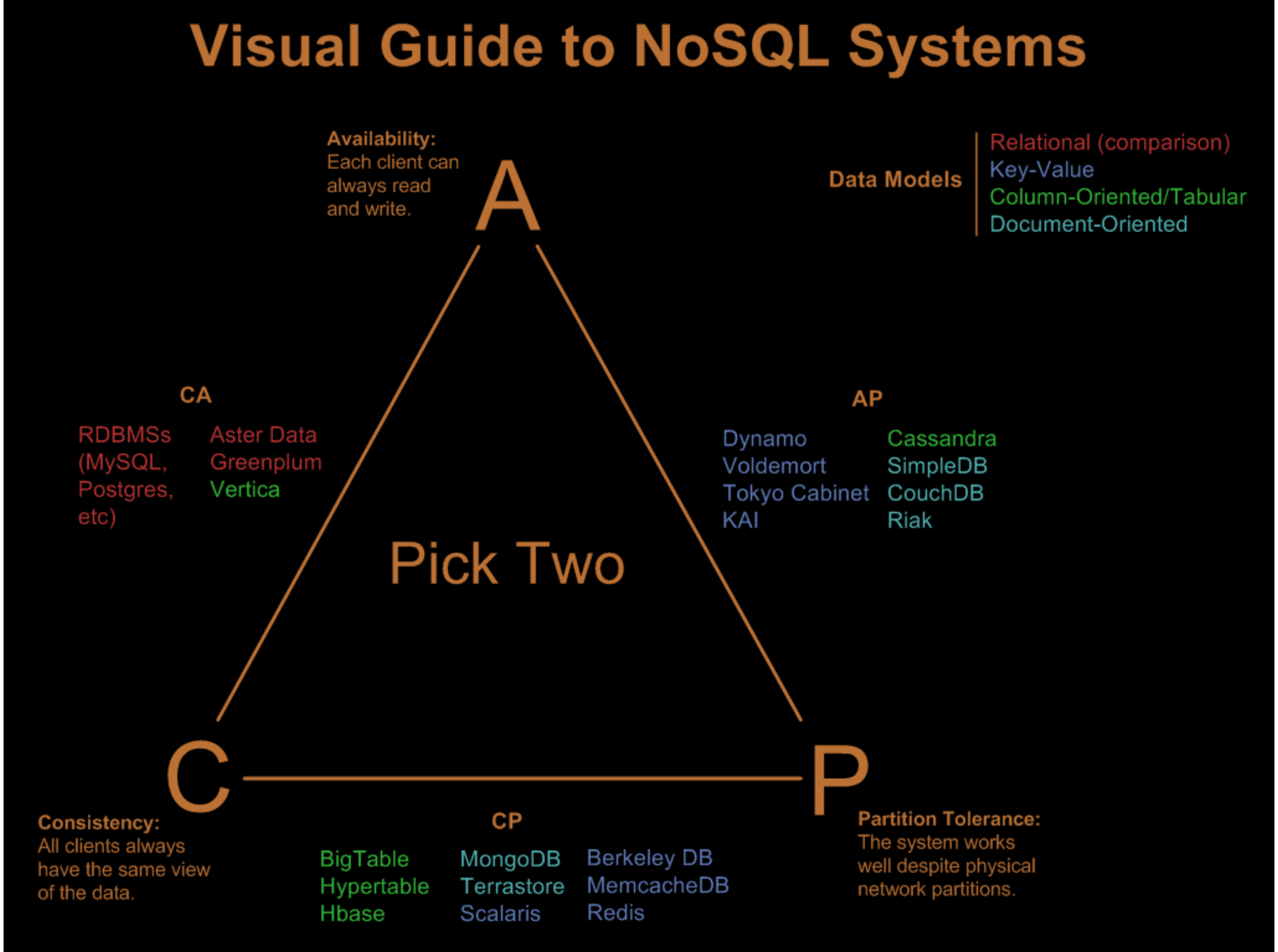
AP: Always provides a "best-effort" response even in presence of network failures (*Eventual consistency*)

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;


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

ACID vs. BASE trade-off

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

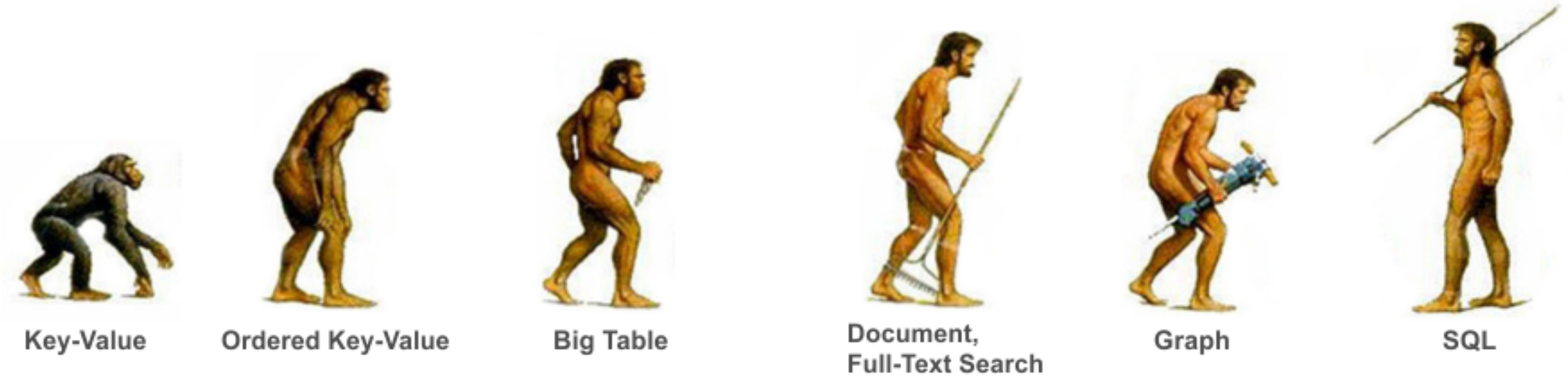
Given **BASE** ’s loose consistency, developers **need to** be more knowledgeable and **rigorous** about **consistent** data if they choose a BASE store for their application.

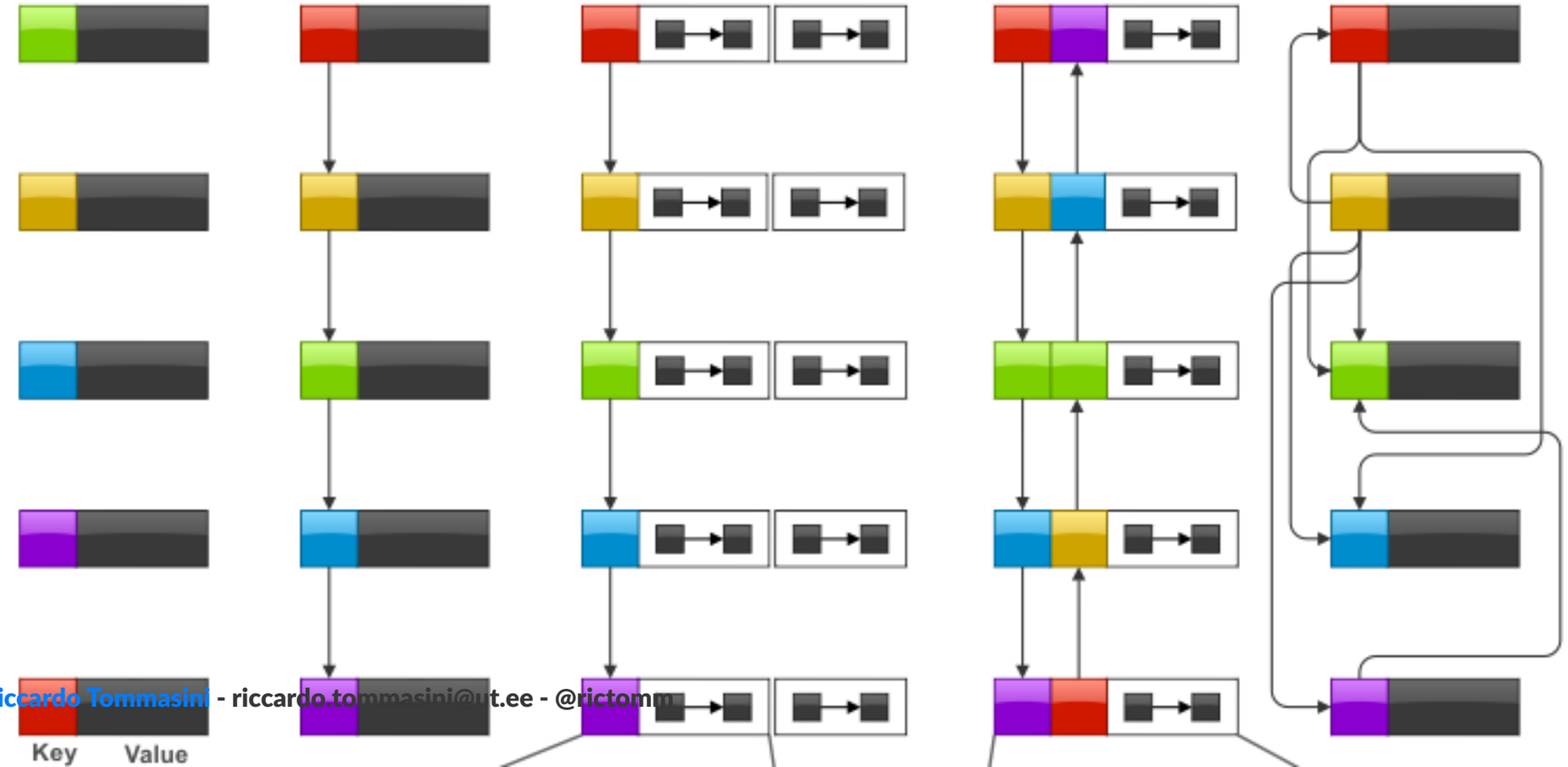
Planning around **BASE** limitations can sometimes be a major **disadvantage** when compared to the simplicity of ACID transactions.

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

History of Data Models[5]

[5] by Ilya Katsov


Key-Value
Ordered Key-Value
Big Table
Document,
Full-Text Search
Graph
SQL


Key
Value

Life beyond Distributed Transactions: an Apostate's Opinion

Position Paper

Pat Helland

Amazon.Com
705 Fifth Ave South
Seattle, WA 98104
USA
PHelland@Amazon.com

The positions expressed in this paper are personal opinions and do not in any way reflect the positions of my employer Amazon.com.

ABSTRACT

Many decades of work have been invested in the area of distributed transactions including protocols such as 2PC, Paxos, and various approaches to quorum. These protocols provide the application programmer a façade of global serializability. Personally, I have invested a non-trivial portion of my career as a strong advocate for the implementation and use of platforms providing guarantees of global serializability.

My experience over the last decade has led me to liken these platforms to the Maginot Line[1]. In general, application developers simply do not implement large scalable applications assuming distributed transactions. When they attempt to use distributed transactions, the projects founder because the performance costs and fragility make them impractical. Natural selection kicks in… Instead, applications are built using different techniques which do not provide the same transactional guarantees but still meet the needs of their businesses.

This paper explores and names some of the practical approaches used in the implementations of large-scale mission-critical applications in a world which rejects distributed transactions. We discuss the management of fine-grained pieces of application data which may be repartitioned over time as the application grows. We also discuss the design patterns used in sending messages between these repartitionable pieces of data.

The reason for starting this discussion is to raise awareness of new patterns for two reasons. First, it is my belief that this awareness can ease the challenges of people hand-crafting very large scalable applications. Second, by observing the patterns, hopefully the industry can work towards the creation of platforms that make it easier to build these very large applications.

[1] The Maginot Line was a huge fortress that ran the length of the Franco-German border and was constructed at great expense between World War I and World War II. It successfully kept the German army from directly crossing the border between France and Germany. It was quickly bypassed by the Germans in 1940 who invaded through Belgium.

3rd Biennial Conference on Innovative DataSystems Research (CIDR) January 7-10, Asilomar, California USA.

1. INTRODUCTION

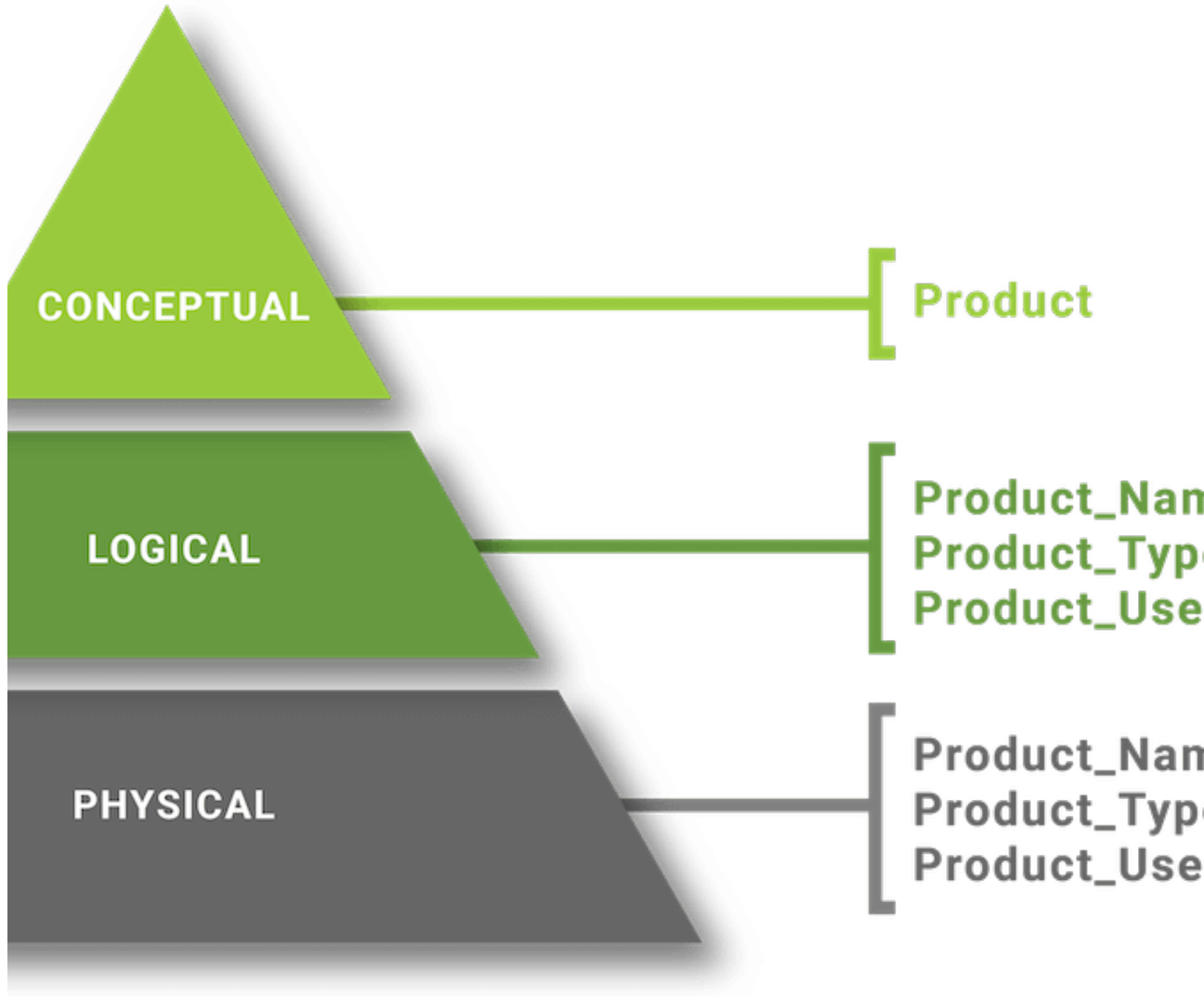
Let's examine some goals for this paper, some assumptions that I am making for this discussion, and then some opinions derived from the assumptions. While I am keenly interested in high availability, this paper will ignore that issue and focus on scalability alone. In particular, we focus on the implications that fall out of assuming we cannot have large-scale distributed transactions.

Goals

This paper has three broad goals:

- Discuss Scalable Applications
  Many of the requirements for the design of scalable systems are understood implicitly by many application designers who build large systems.

Shall we rethink the three-layered modeling?



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._

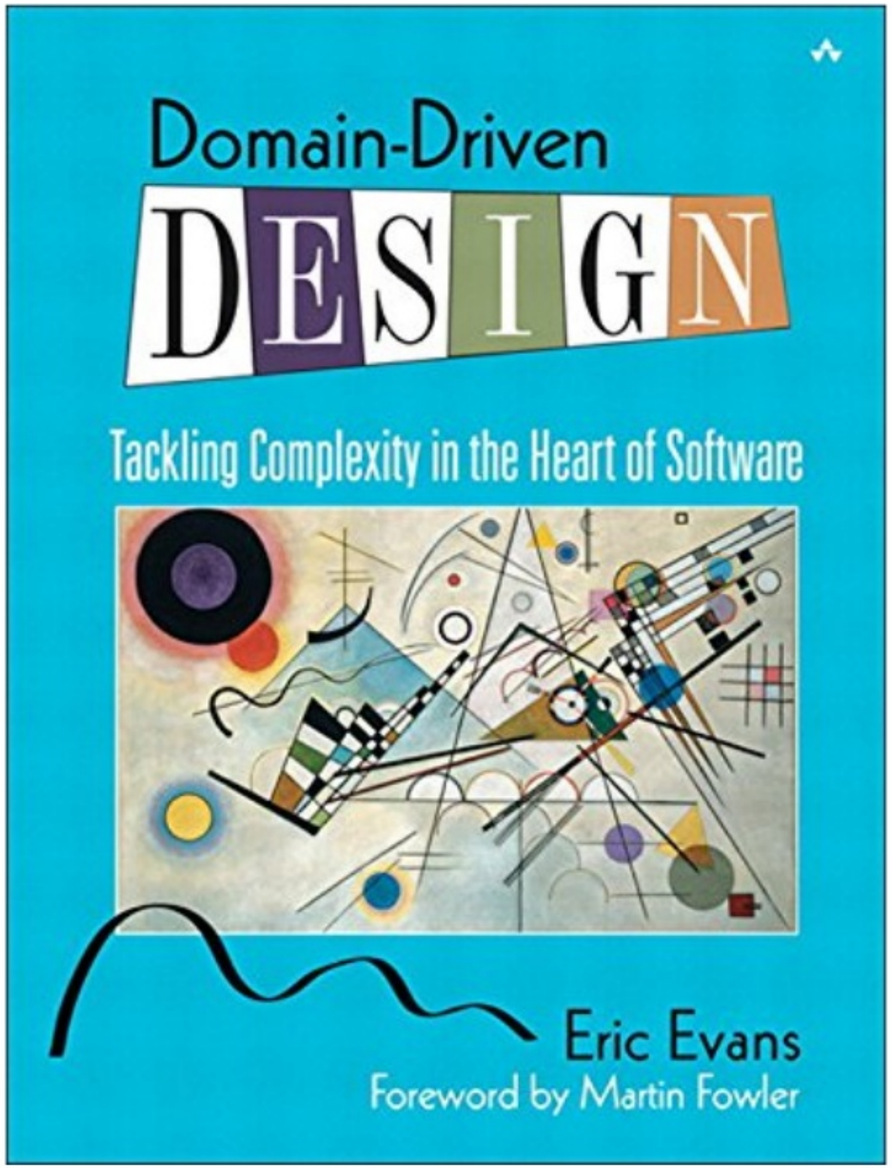
Domain Driven Design[68]

Domain-Driven Design is a **language**- and **domain-centric** approach to software design for complex problem domains.

DDD promotes the reduction of the translation cost between business and technical terminology by developing an **ubiquitus language** that embeds domain terminology into the software systems.

DDD consists of a collection of **patterns**, **principles**, and **practices** that allows teams to **focus on** the core **business** goals while **crafting** software.

intro



[68] book

Domain Driven Design[68]

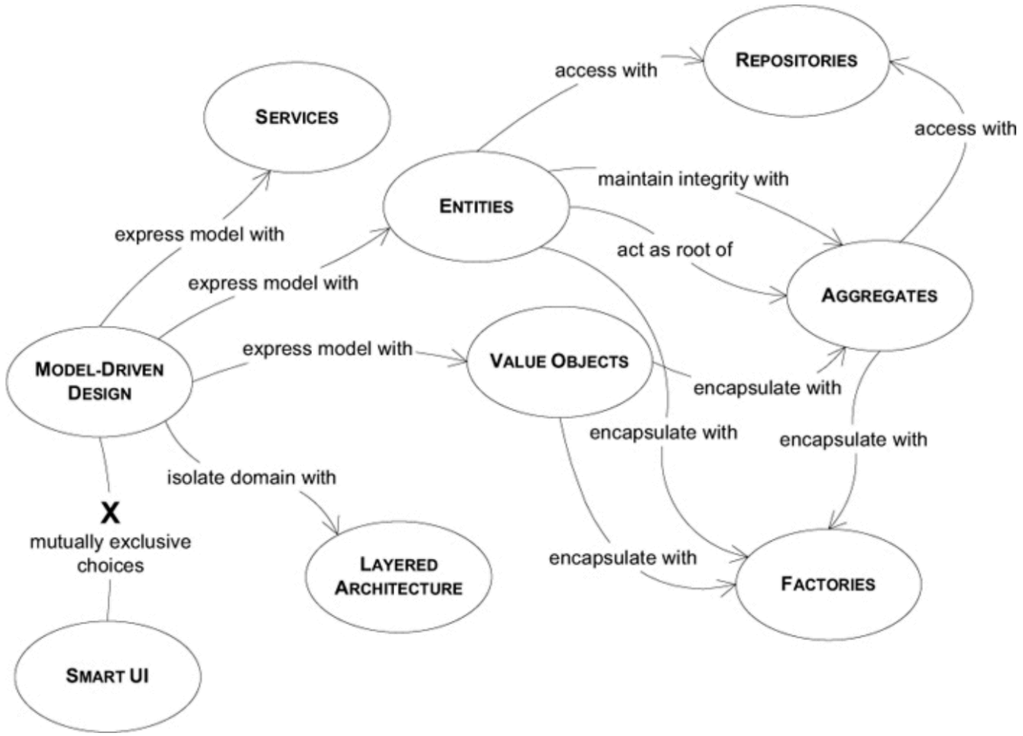
Domain-Driven Design is a **language**- and **domain-centric** approach to software design for complex problem domains.

DDD promotes the reduction of the translation cost between business and technical terminology by developing an **ubiquitus language** that embeds domain terminology into the software systems.

DDD consists of a collection of patterns, principles, and practices that allows teams to focus on the core t business goels while crafting software.

[68] book

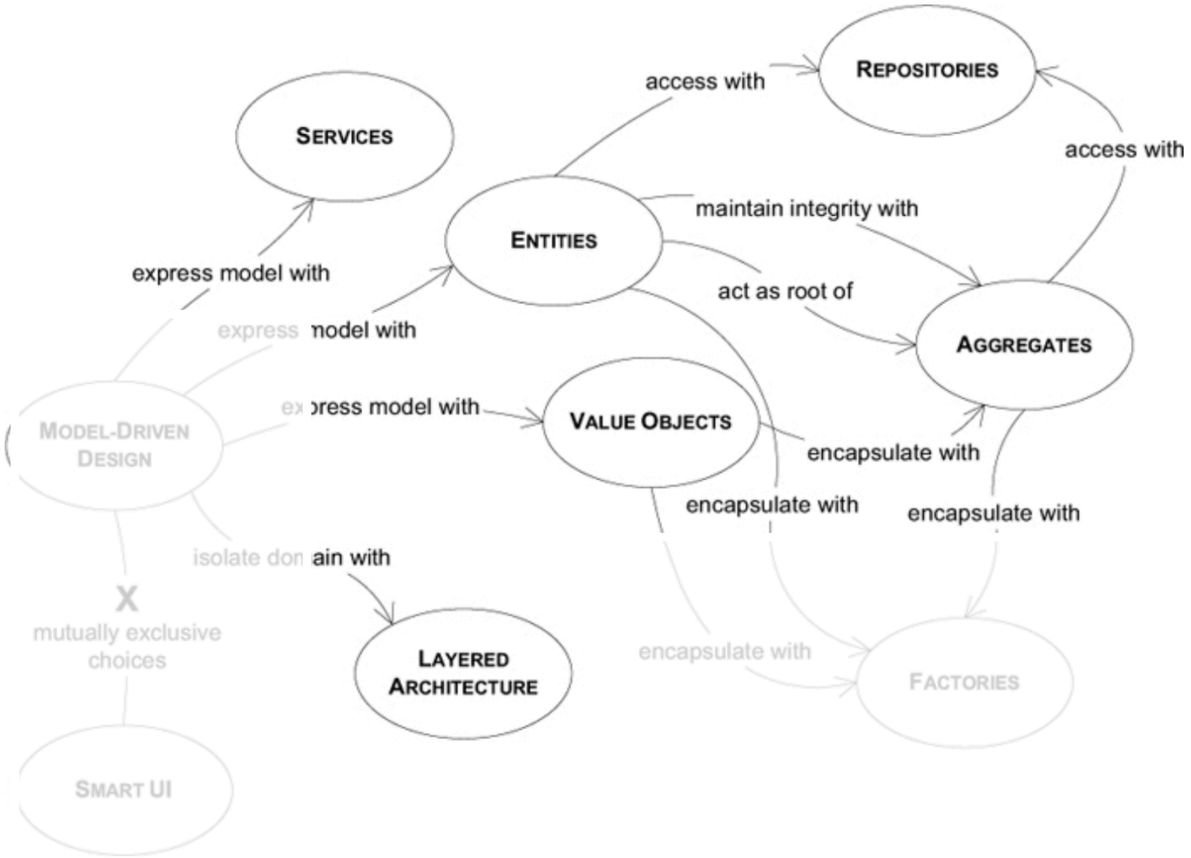
Domain Driven Design



A navigation map of the language of MODEL-DRIVEN DESIGN

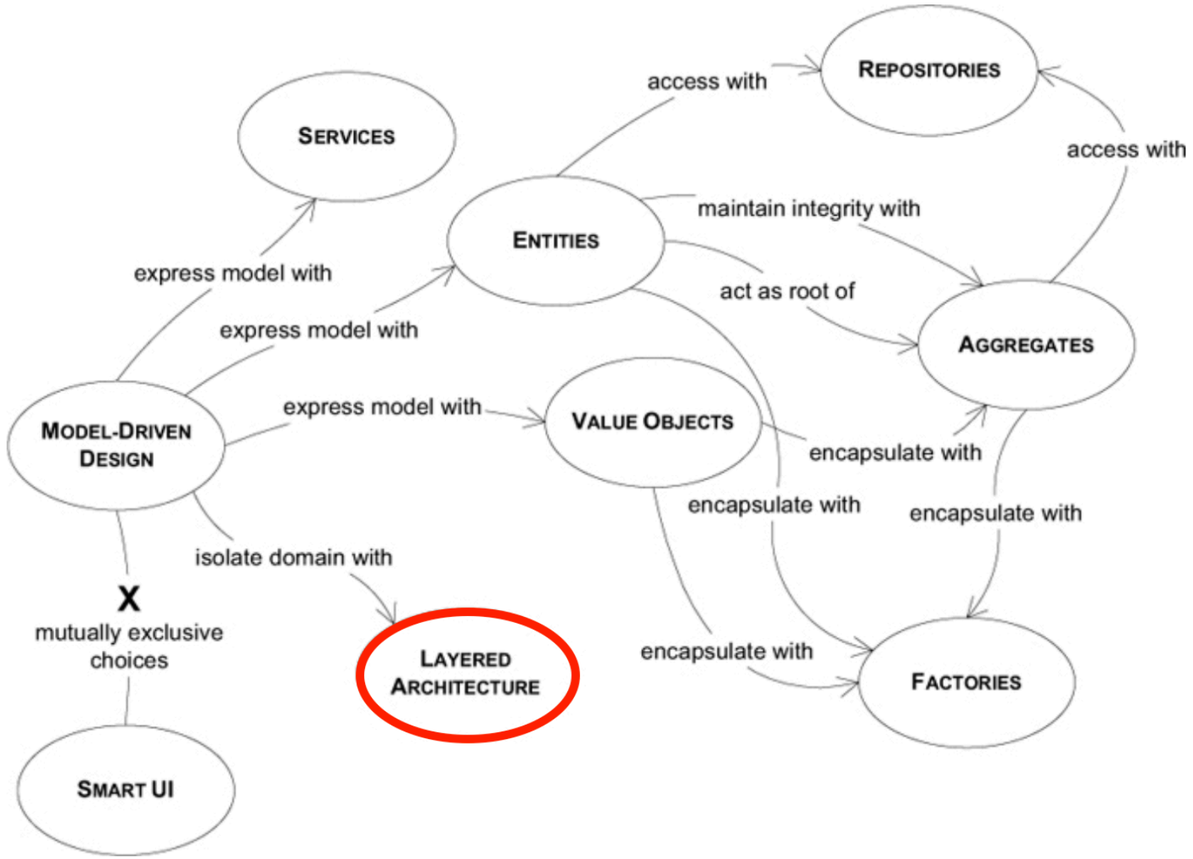
source

Domain Driven Design



A navigation map of the language of MODEL-DRIVEN DESIGN

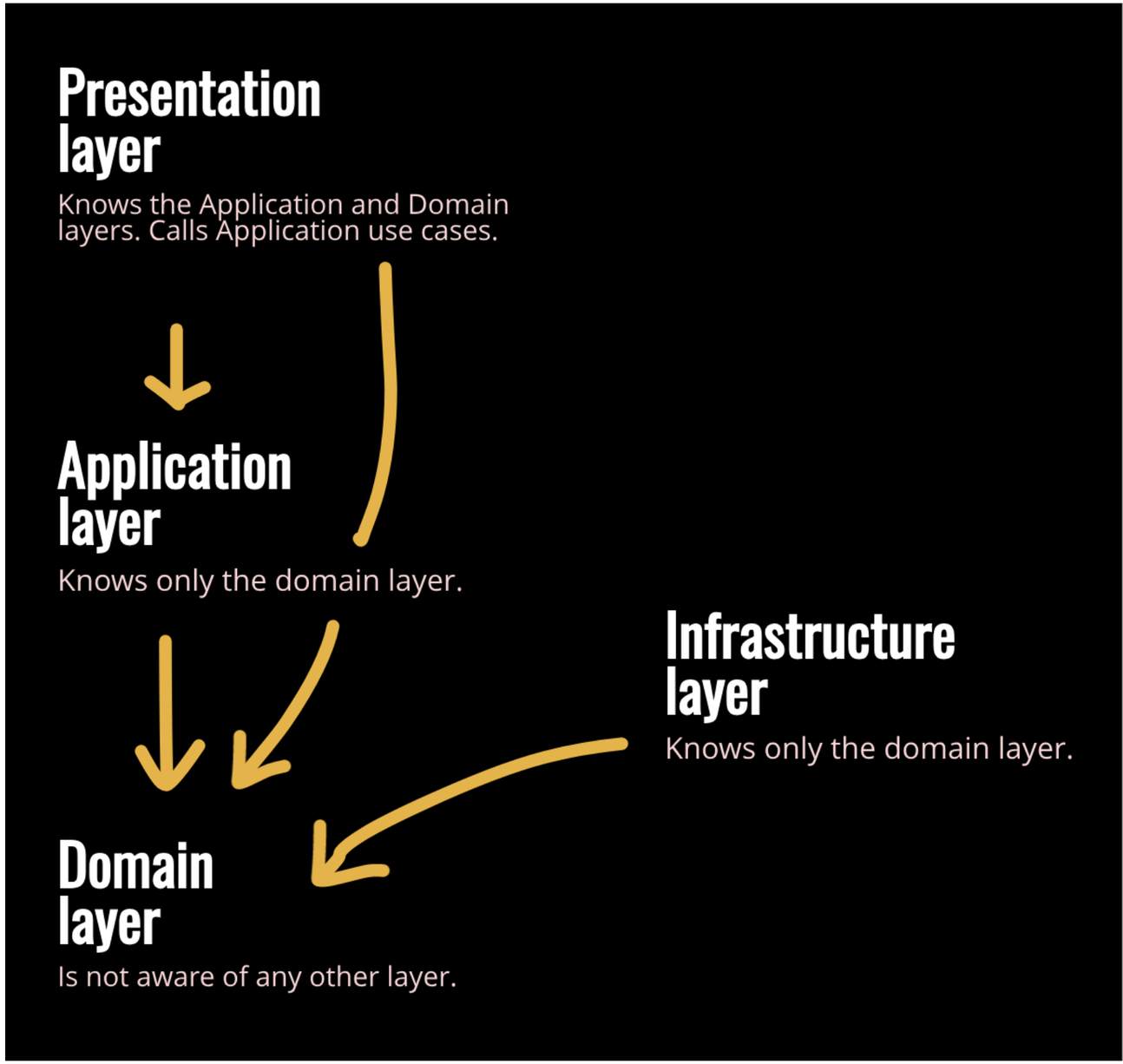
Domain Driven Design



A navigation map of the language of MODEL-DRIVEN DESIGN

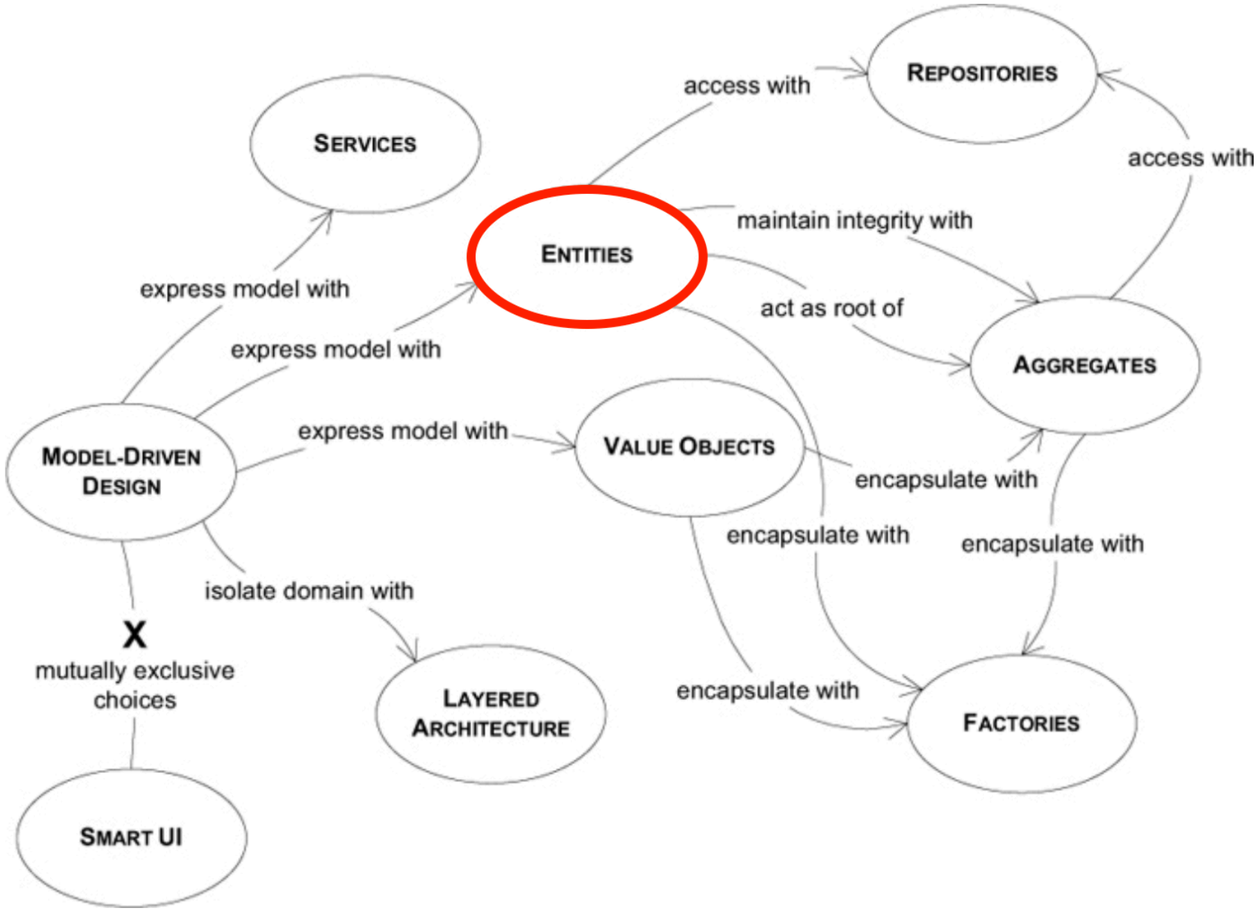
The Layered Architecture

Layer	Description
Presentation Layer	Responsible for showing information to the user and interpreting the user's commands.
Application Layer	Defines the jobs the software is supposed to do and directs the expressive domain objects to work out problems
Domain Layer	Responsible for representing concepts of the business, information about the business situation, and business rules.
Infrastructure Layer	Provide generic technical capabilities that support the higher layers: message sending for the application, persistence for the domain, drawing widgets for the UI, etc.



Entities

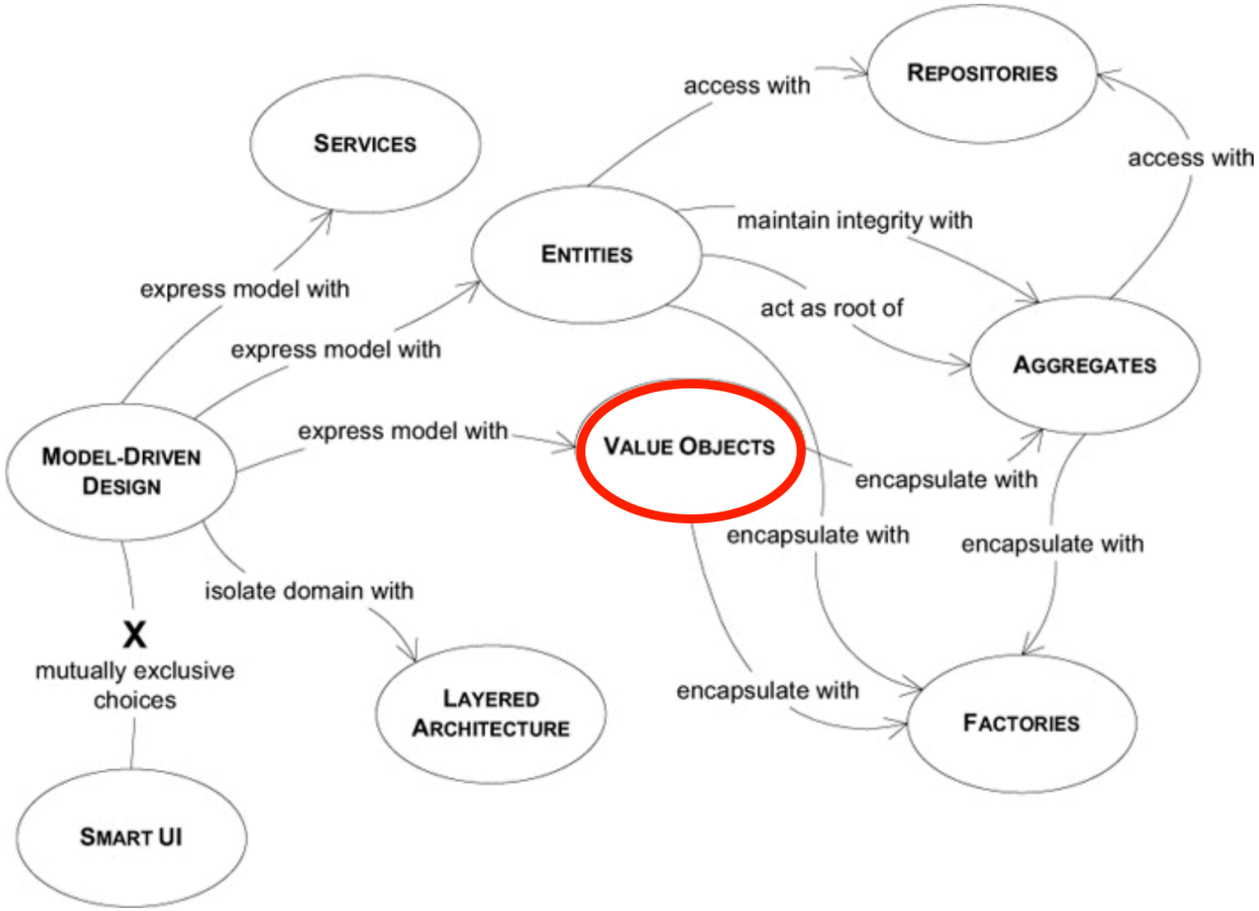
- Are objects defined primarily by their identity
- Their identities must be defined so that they can be effectively tracked. We care about *who* they are rather than *what* information they carry
- They have lifecycles may can radically change their form and content, while a thread of continuity must be maintained.
- E.g., bank accounts, deposit transaction.



A navigation map of the language of MODEL-DRIVEN DESIGN

Value Objects

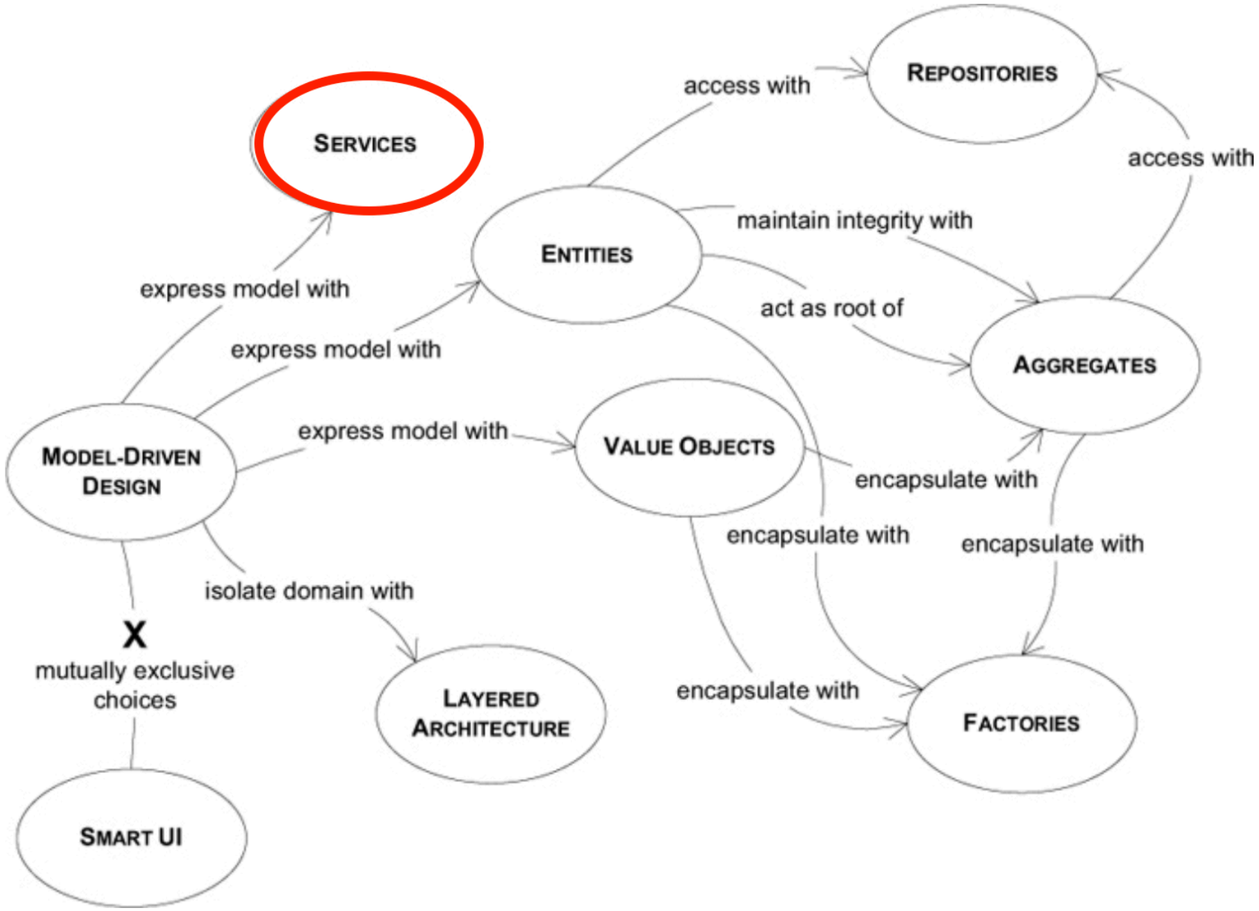
- Value Objects represent a descriptive aspect of the domain that has no conceptual identity.
  - They are instantiated to represent elements of the design that we care about only for *what* they are, not *who* they are.
  - E.g., For example, street, city, and postal code shouldn't be separate attributes of a Person object.



A navigation map of the language of MODEL-DRIVEN DESIGN

Services

- Services are operations offered as an interface that stands alone in the model, without encapsulating state as Entities and Value Objects do.
  - They are a common pattern in technical frameworks, but they can also apply in the domain layer.
  - The name “service” is ment to emphasize the relationship with other objects.



A navigation map of the language of MODEL-DRIVEN DESIGN

The Lifecycle of a Domain Object

Every object has a lifecycle. It is **born**, it may go **through** various **states**, it eventually is either **archived** or **deleted**.

The problems fall into two categories:
- **Maintaining integrity** throughout the lifecycle
- **Preventing** the model from getting swamped by the **complexity** of managing the lifecycle.

Aggregates and Repositories

The most important concepts for this are Aggregates and Repositories[63]

Aggregates are a cluster of Entities and Value Objects that make sense domain-wise and are retrieved and persisted together.

E.g. A Car is an aggregate of wheel, engine, and the customer

Repositories offer an interface to retrieve and persist aggregates, hiding lower level details from the domain.

E.g. Sold cars catalogue

[63] an Aggregate is always associated with one and only one Repository.

Event Sourcing[64]

- The fundamental idea of Event Sourcing is ensuring that every change to the state of an application is captured in an event object,
- Event objects are immutable and stored in the sequence they were applied for the same lifetime as the application state itself.

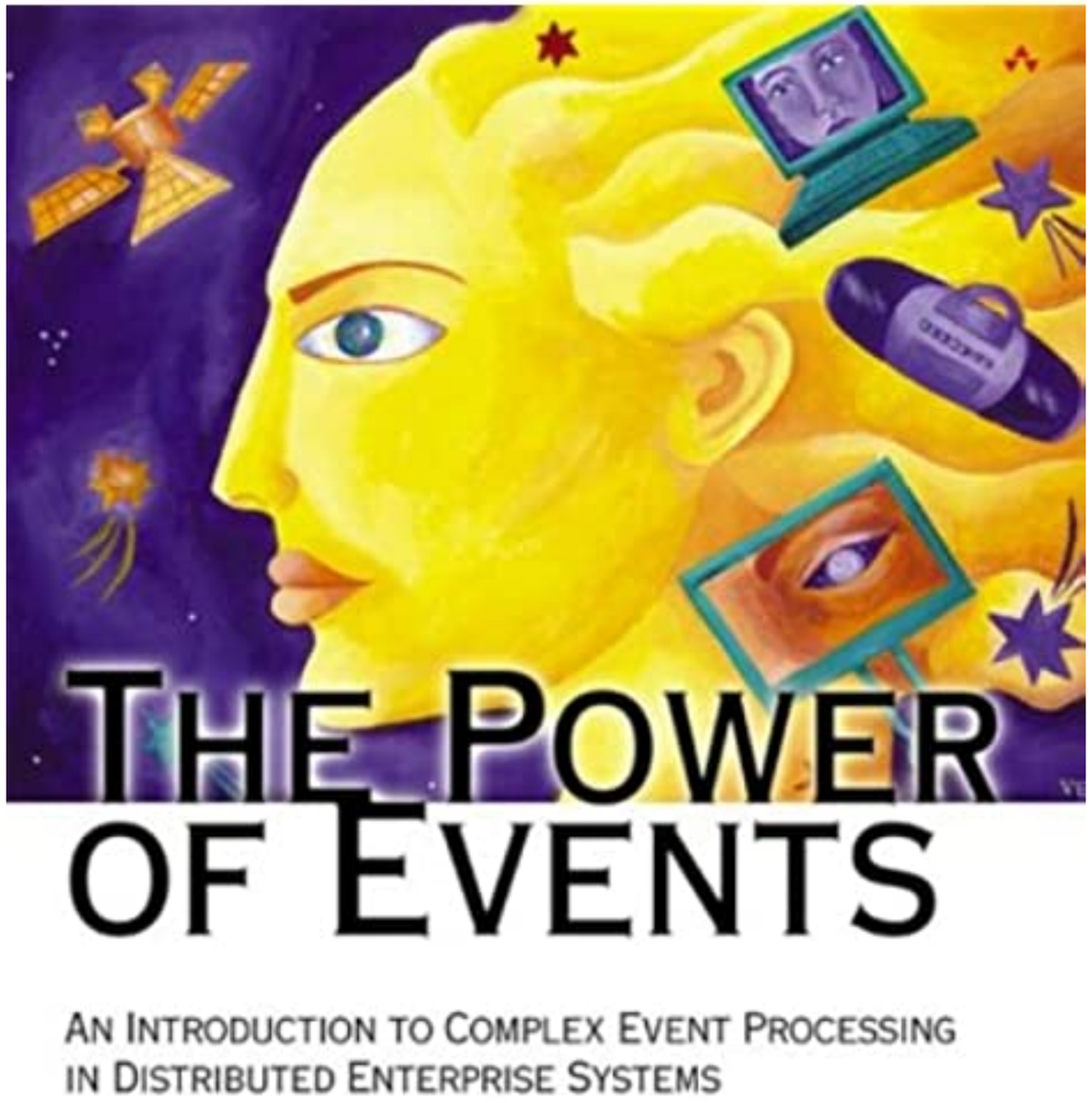
[64] Martin Fowler, link

The Power of Events

Events are both a **fact** and a **notification**.

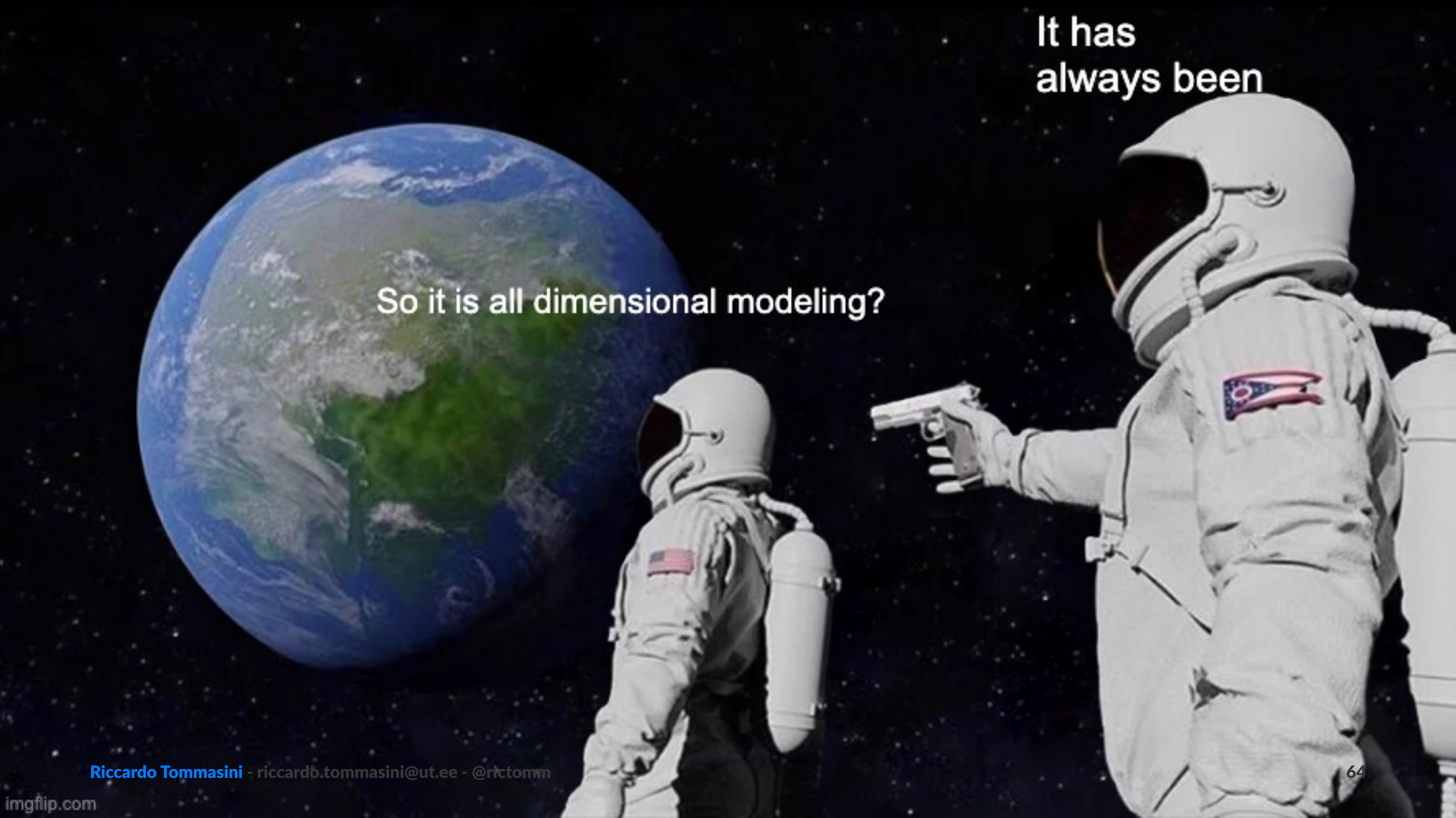
They represent **something** that **happened** in the **real world** but include no expectation of any future action.

They **travel** in only **one direction** and expect no response (sometimes called “fire and forget”), but one **may be “synthesized” from a subsequent event**.




Hey, I've seen this one!


What do you mean you've seen it?
It's brand new!


It has
always been
So it is all dimensional modeling?
imgflip.com