Research Project Proposal: Towards a unifying model for data-intensive applications

Nicolò Felicioni
nicolo.felicioni@mail.polimi.it
Computer Science and Engineering

POLITECNICO MILANO 1863

HONOURS PROGRAMME HP-SR in Information Technology

Data-intensive applications

What is a «data-intensive» application?

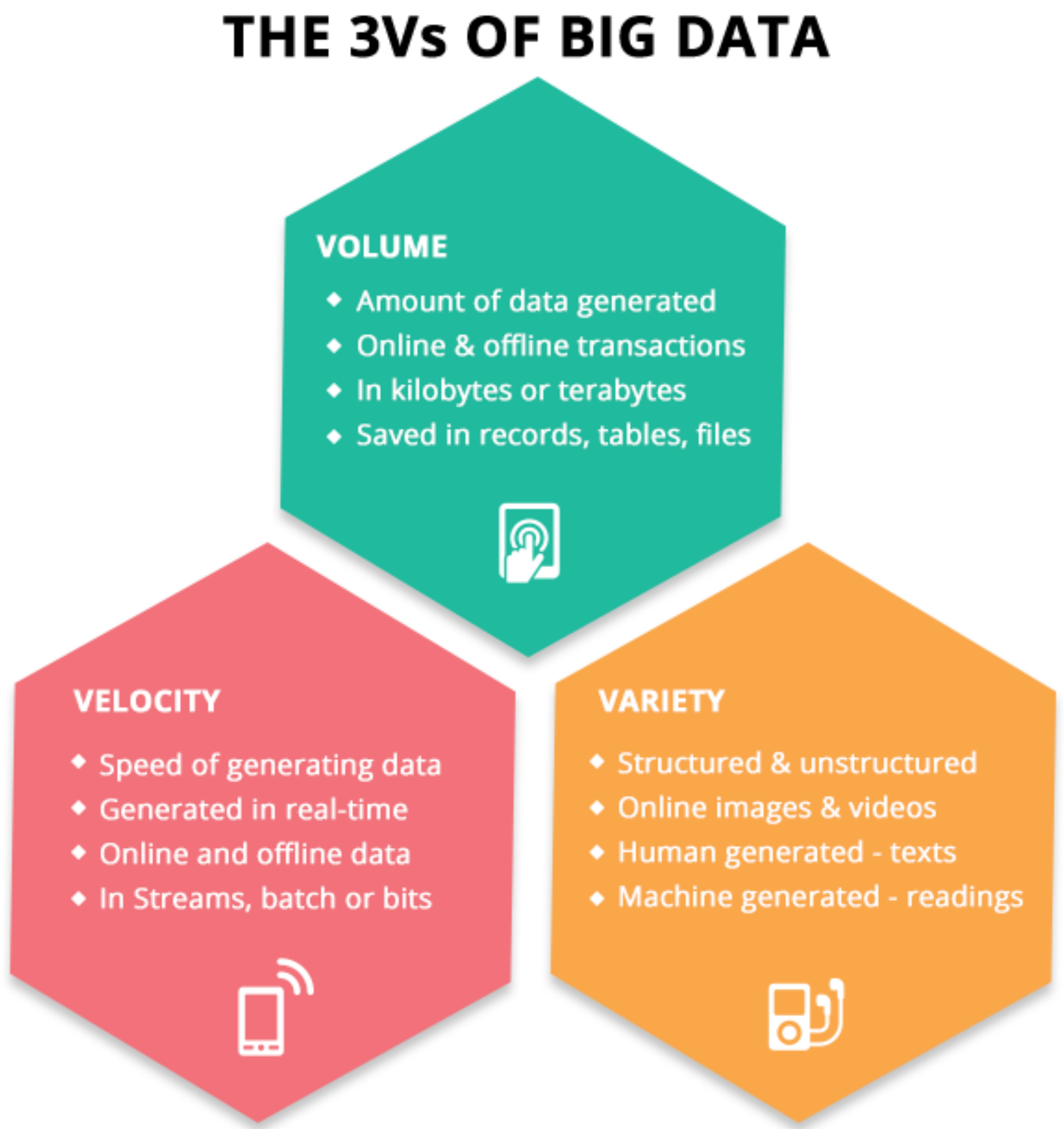
We are talking about:

- Software applications
- Big data

The Big Data era

Big data means (at least) three problems:

1. Big Volume
2. Big Velocity
3. Big Variety



Data-intensive vs. Compute-intensive

Data-intensive application:
  data (the quantity, the speed at which it is changing, the variety) is the primary challenge

Compute-intensive application:
  CPU is the bottleneck

A tale of two worlds

Now a step back into the state of the art

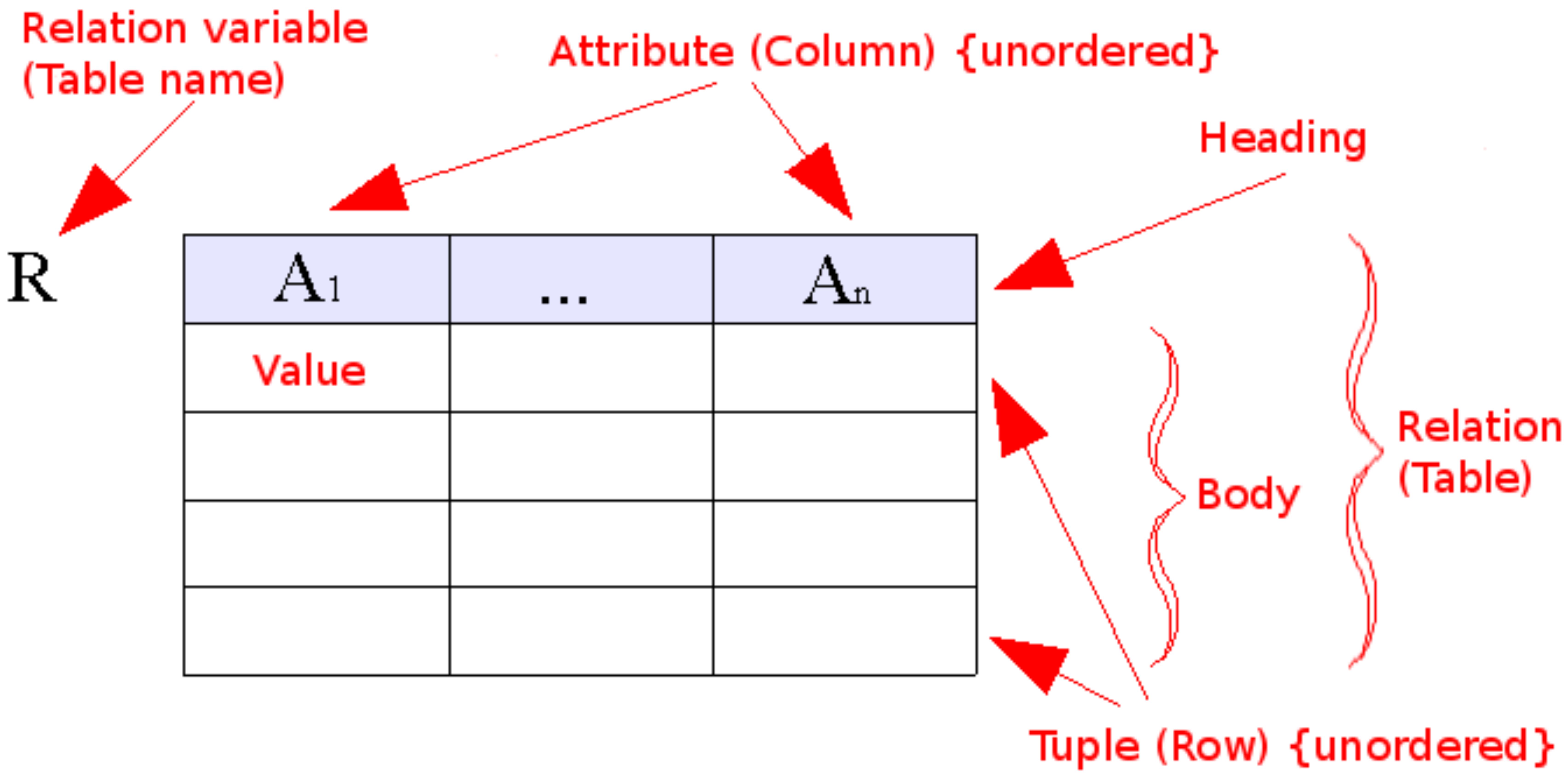
Two main areas:

- Database research area
- Distributed systems research area

Database basics

- Collection of data
- Software used to manage databases is called Database Management System (DBMS)
- The first data model was the relational model

Relational model

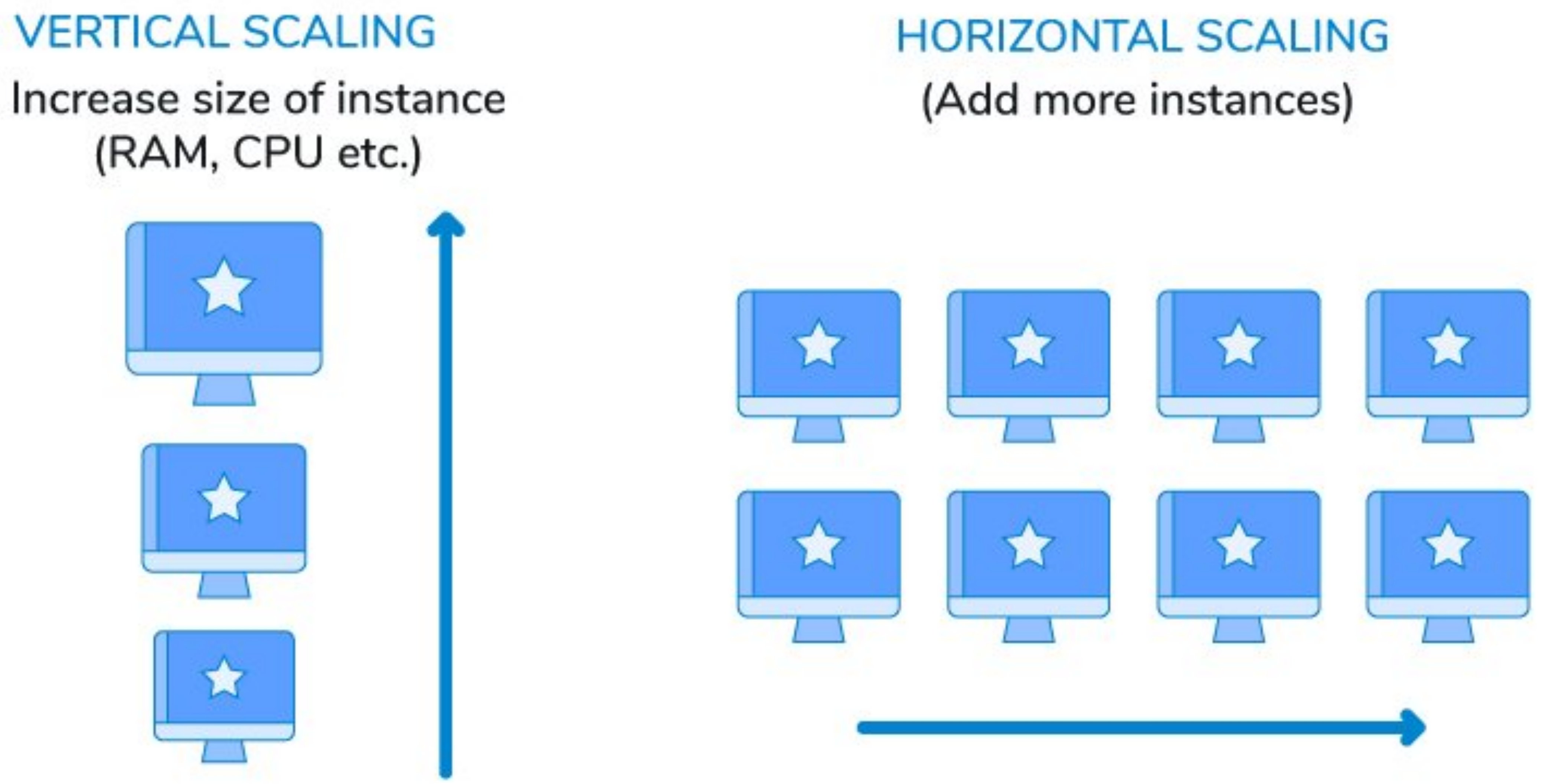


Transactions

- Classical DBMSs usually support transactions
- A transaction is a unit of work that must be Atomic, Consistent, Isolated and Durable (ACID)
- On-line Transaction Processing (OLTP) is a scenario where a database is used mainly for processing multiple transactions
- The transaction management can be a bottleneck when implemented in data-intensive systems

Issues with early databases

- Classical solutions (Oracle, MySQL) were not good at “horizontal” scaling



- A new type of systems called NoSQL started to gain relevance in the 2000s

Image from:https://www.redswitches.com/hs-fs/hubfs/scaling-image.jpg?width=1580&name=scaling-image.jpg

NoSQL

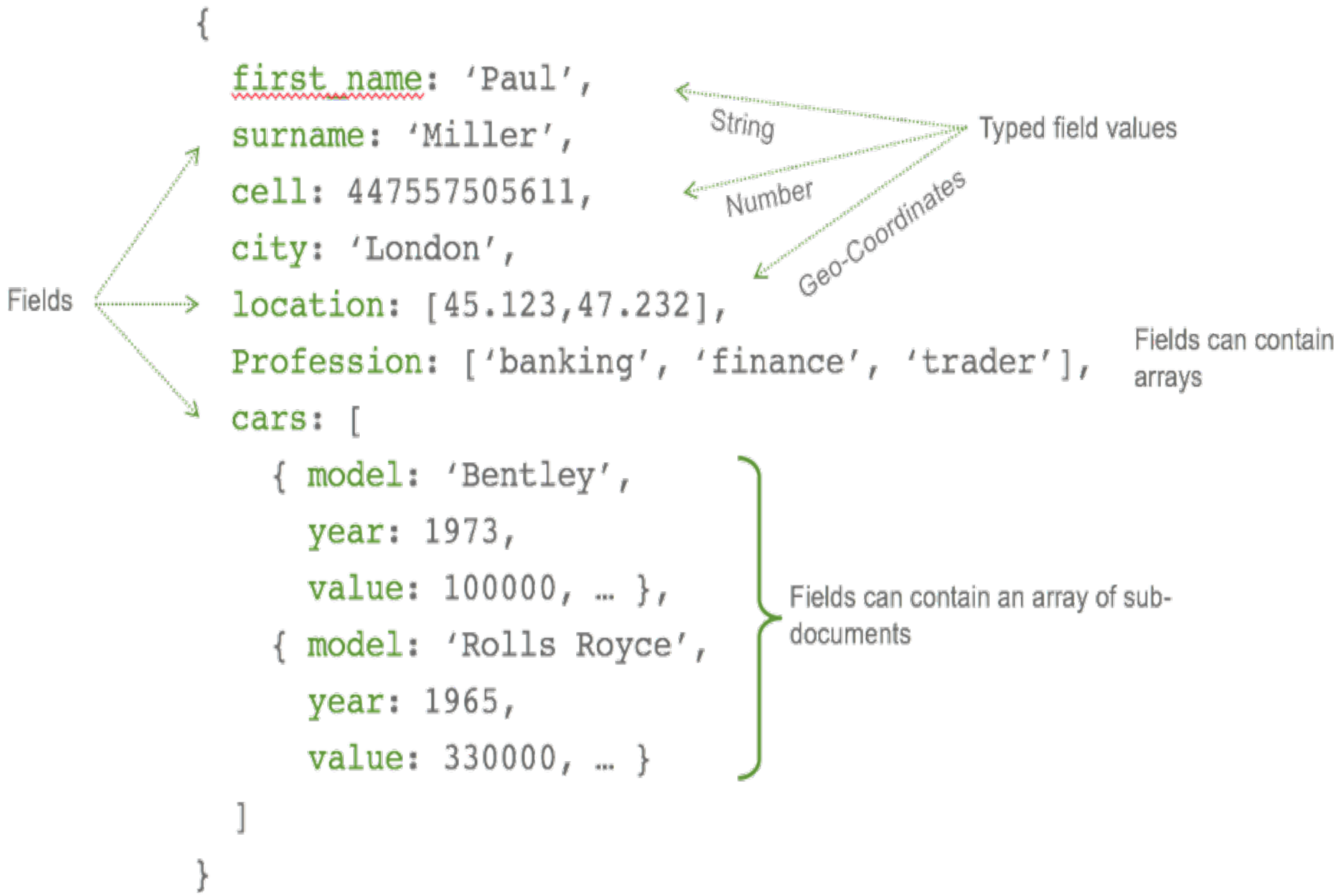
NoSQL started for data-intensive needs – Volume, Variety

Usually a NoSQL database is:

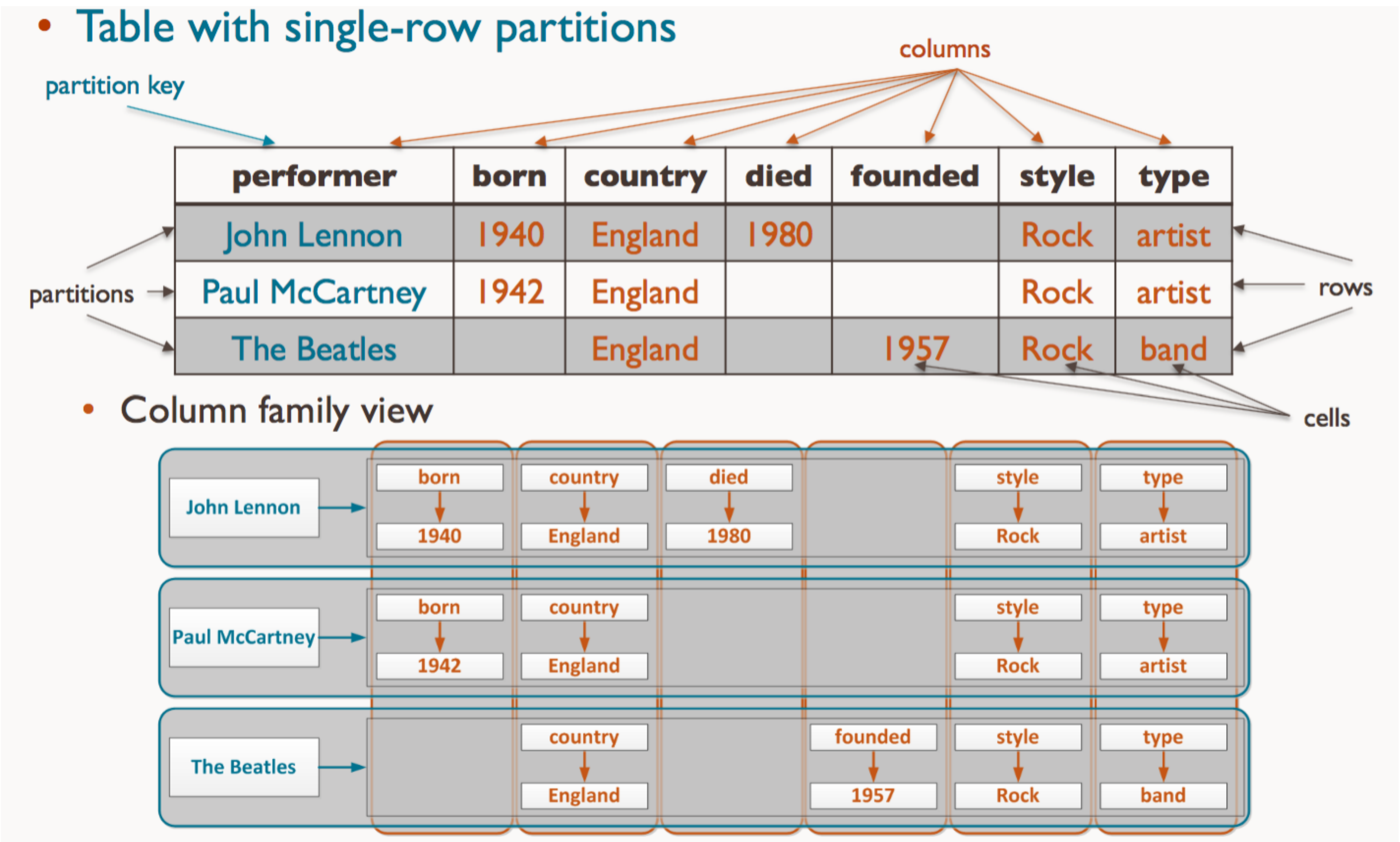
- non-relational
- distributed
- open-source
- horizontally scalable

NoSQL data models

- NoSQL is an inherently heterogeneous category



Document data model (e.g. MongoDB)



Wide column data model (e.g. Cassandra)

Images from https://studio3t.com/knowledge-base/articles/nosql-database-types/#key-value-data-store

NoSQL issues

- NoSQL systems are valuable tools, especially for data-intensive requirements
- Though they have a big flaw: lack of support for full ACID transactions
- And OLTP market is still relevant

NewSQL

- The solution for scalable OLTP scenarios: NewSQL
- They try to make scalable as much as possible the traditional relational systems, while preserving all their guarantees
- Different approaches were adopted to implement transactions with strong consistency and isolation with sufficient performance and availability

NewSQL approaches

- Synchronization based on specialized hardware like atomic clocks, adopted by Google Spanner
- Limit transaction expressivity, adopted by Calvin
- Using information on replication provided explicitly by the user the to optimize transactions in distributed settings, adopted by VoltDB

A tale of two worlds

Two main areas:

- Database research area
- Distributed systems research area

MapReduce

- In Distributed Systems research, systems explicitly designed for distributed processing in large-scale compute infrastructures started to gain popularity
- These systems trace their roots to Google's programming model called MapReduce (2004)

MR fundamentals

- The computation is split into two phases, Map and Reduce
- **Map** processes individual elements
  For each of them outputs one or more <key, value> pairs
- **Reduce** processes all the values with the same key and outputs a value
- The runtime system controls scheduling, load balancing, communication, fault tolerance

From the slides of A. Margara from Distributed Systems course

MR word count example

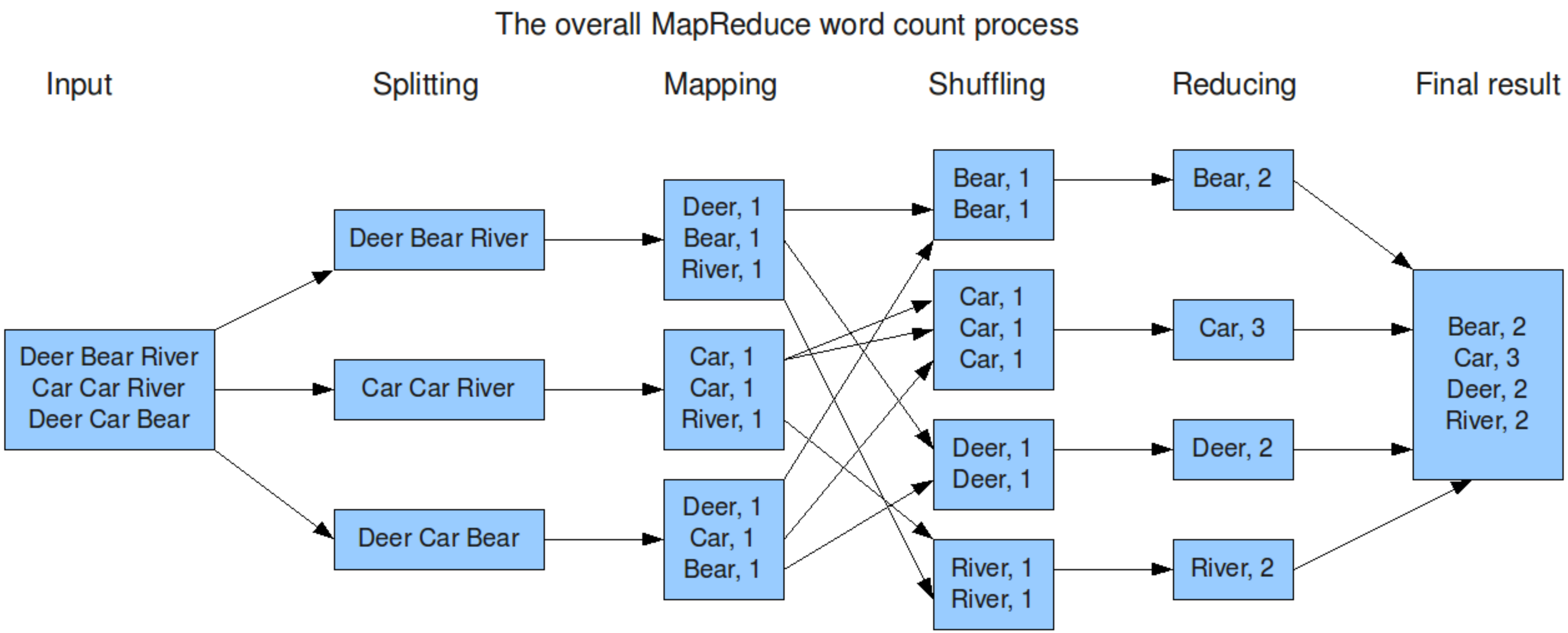


Image from: https://blog.trifork.com/wp-content/uploads/2009/08/MapReduceWordCountOverview1.png

Beyond MapReduce

In the last decade, many systems extended and improved the MapReduce abstraction in many ways

- From two processing steps to arbitrary acyclic graphs of transformations
- From batch processing to stream processing
- From disk to main-memory or hybrid approaches

Examples:

- Apache Spark for batch processing
- Apache Flink for stream processing

Batch processing - Spark

- Similar to MapReduce
  - Instead of only two stages (map and reduce) …
  - … arbitrary number of stages
- Intermediate results can be cached in main memory if they are reused multiple times
- Scheduling of tasks (stages) ensures that the computation takes place close to the data

From the slides of A. Margara from Distributed Systems course

Stream processing - Flink

- A job is not split into stages that are scheduled
- Instead, all the operators are instantiated as soon as the job is submitted
    - They communicate using TCP channels
    - An operator can start processing as soon as it has some data available from the previous ones
        - Pipeline architecture where multiple operators are simultaneously running

From the slides of A. Margara from Distributed Systems course

Data-intensive issues

- The presented data systems –relational DB, NoSQL, NewSQL, MR, batch/stream processing– offer solutions to solve specific data processing and management tasks
- But often requirements of a data-intensive application can be heterogeneous
- Therefore they cannot be satisfied by any of these systems alone

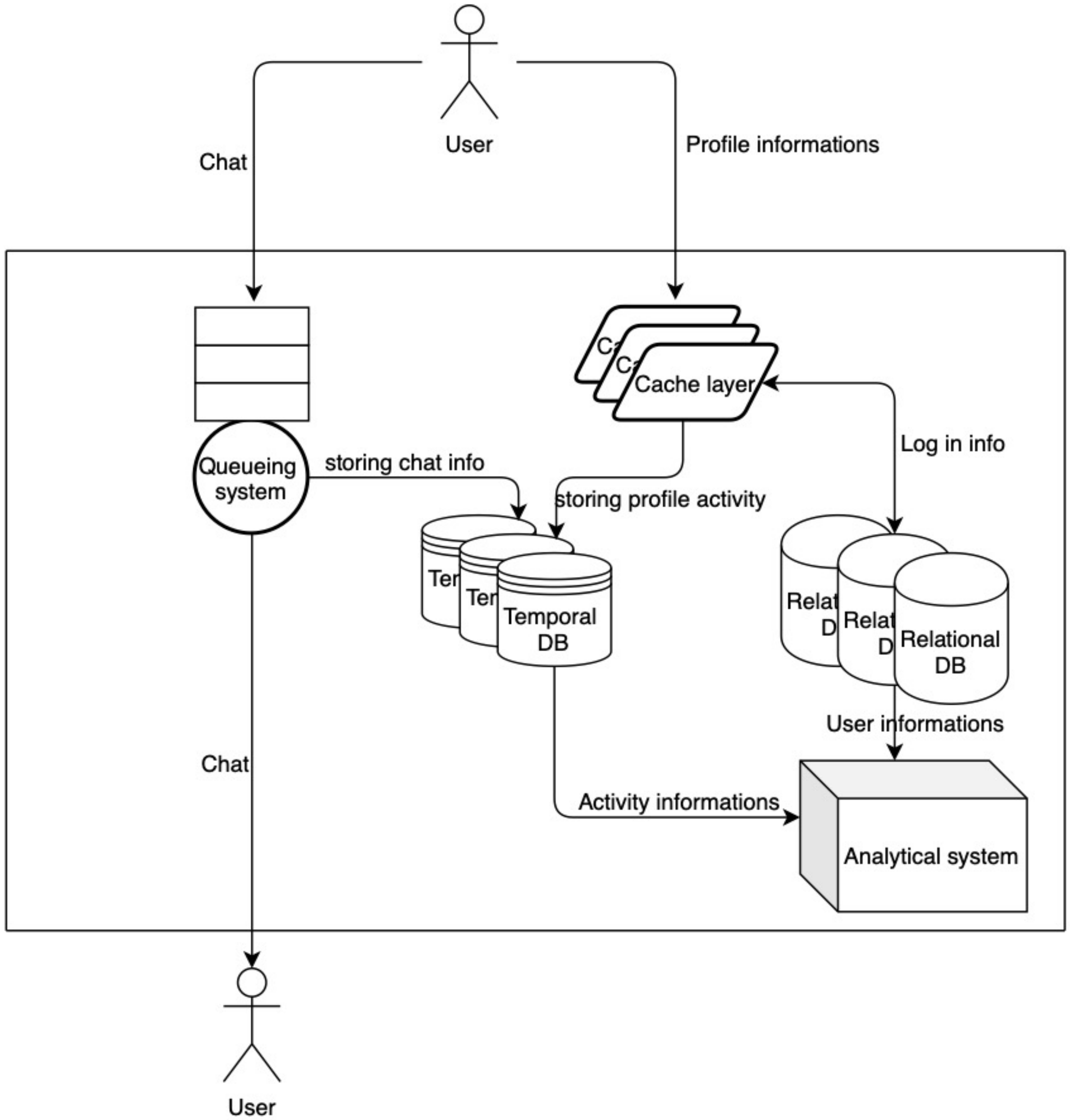
Current approach

- Developers in practice build complex architectures that combine multiple systems
- They implement application logic in order to orchestrate their interaction

Current problems

- In doing so, they lose the benefits provided by the systems in terms of guarantees on the data and transparent deployment and communication
- Also, integrating data systems together necessitates a deep understanding of:
    - Semantics
    - Workload assumptions
    - Performance characteristics
    - Deployment strategies
    - Configuration opportunities

An online collaboration tool example



Developers need to :

- configure individual subsystems
- manually integrate the subsystems
- implement the mechanisms that ensure correctness criteria (profile information is consistent across replicas, temporal database and the queuing system have consistent order of messages, …)
- Take care of performance concerns

A unifying model

- The goal of the research is finding a formal model that defines high-level notions and structures

- The purpose is twofold:

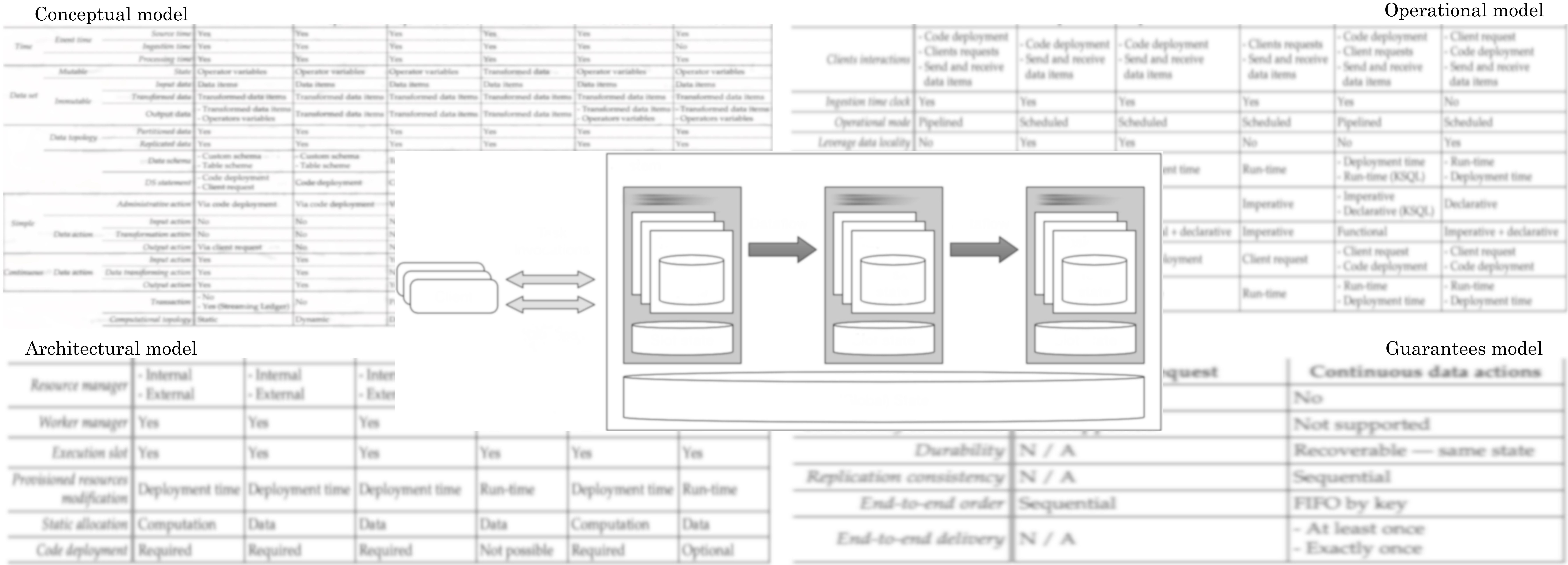
1. the various data-intensive systems usually present intersections among them, therefore a unifying model can be useful to better understand the semantics of the converging concepts of different systems

2. this modeling framework can be a first fundamental step in the direction of a change of paradigm, that leads to a new approach for designing data-intensive application

A unifying model

- In this way, developers no more have to deal with trying to put different and independently developed systems together in a sort of "software collage", where the formal guarantees provided by the single systems could be lost.

A unifying model



Research activity

1. Scope definition

Research activity

1. Scope definition
2. Systems identification and classification

VOLTDB

Flink

cassandra

mongoDB®

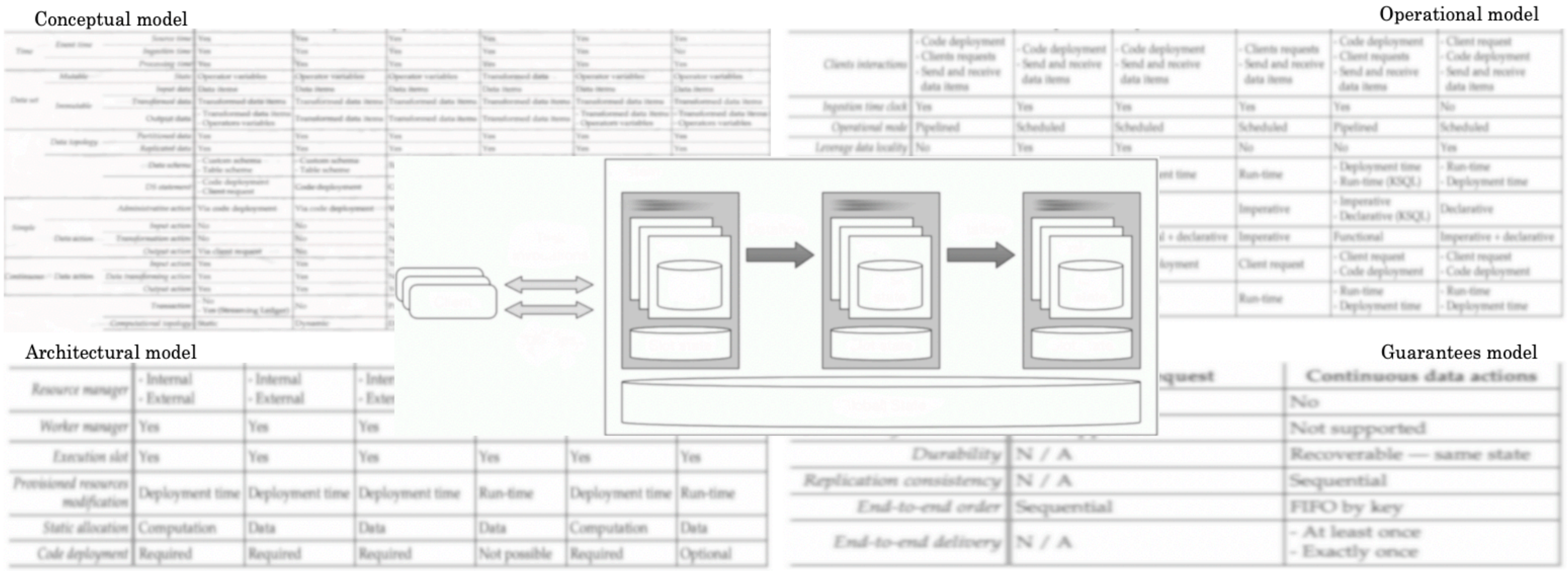
APACHE Spark™

Research activity

1. Scope definition
2. Systems identification and classification
3. Preliminary study of the tools

Research activity

1. Scope definition
2. Systems identification and classification
3. Preliminary study of the tools
4. First model



Research activity

1. Scope definition
2. Systems identification and classification
3. Preliminary study of the tools
4. First model
5. Experiments and consolidated model (iterative task)

Research activity

1. Scope definition
2. Systems identification and classification
3. Preliminary study of the tools
4. First model
5. Experiments and consolidated model (iterative task)
6. Writing

Research activity