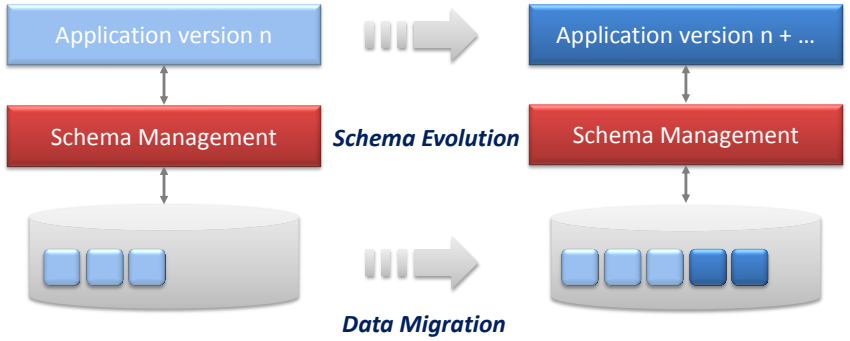
NoSQL Schema Evolution Management and Big Data Migration in the Cloud

Uta Störl

Darmstadt University of Applied Sciences

Joint work with Meike Klettke (University of Rostock) and Stefanie Scherzinger (OTH Regensburg)



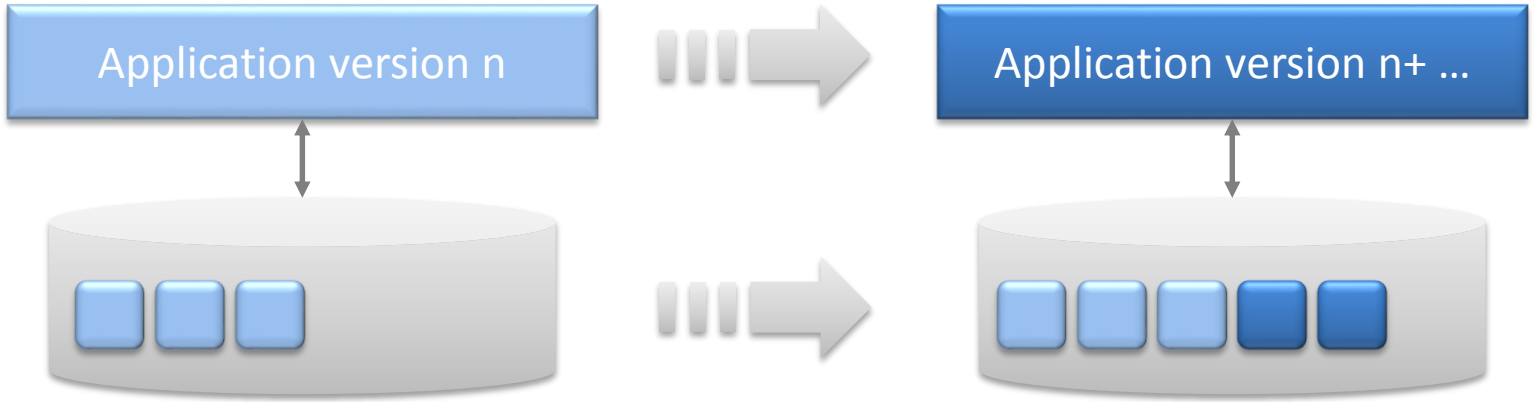
June 2019

Agenda

- Motivation
- NoSQL Schema Evolution Management
- Big Data Migration in the Cloud

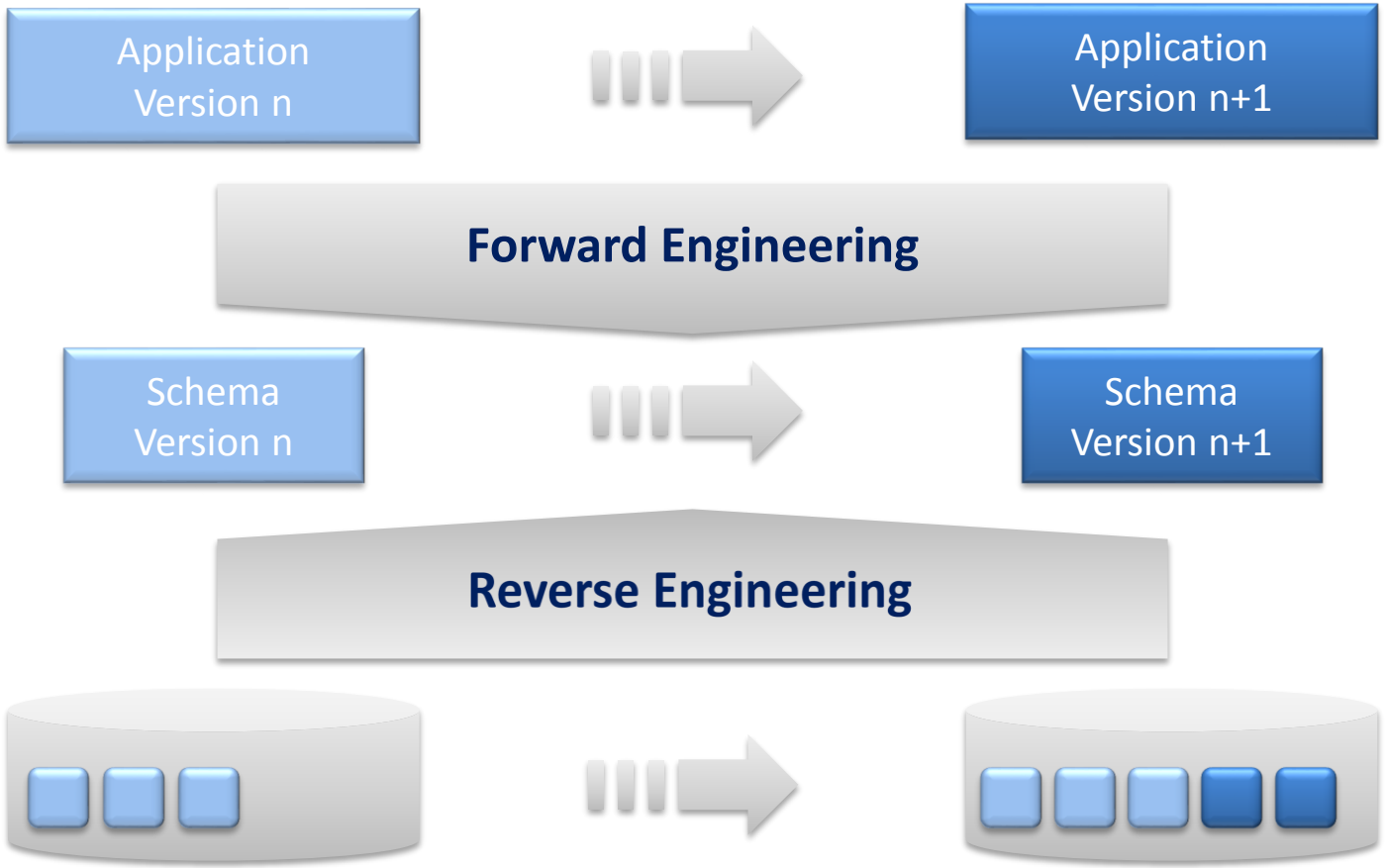
Motivation

- Agile software development with frequent schema changes (weekly up to daily!) ⇨ Schema-flexible NoSQL databases ☺



- However, how to migrate variational data in the productive database?
  - State of the art: Within the application code ☹
  - ⇨**Optional schema management for NoSQL database systems necessary!**

Schema Management for NoSQL Databases



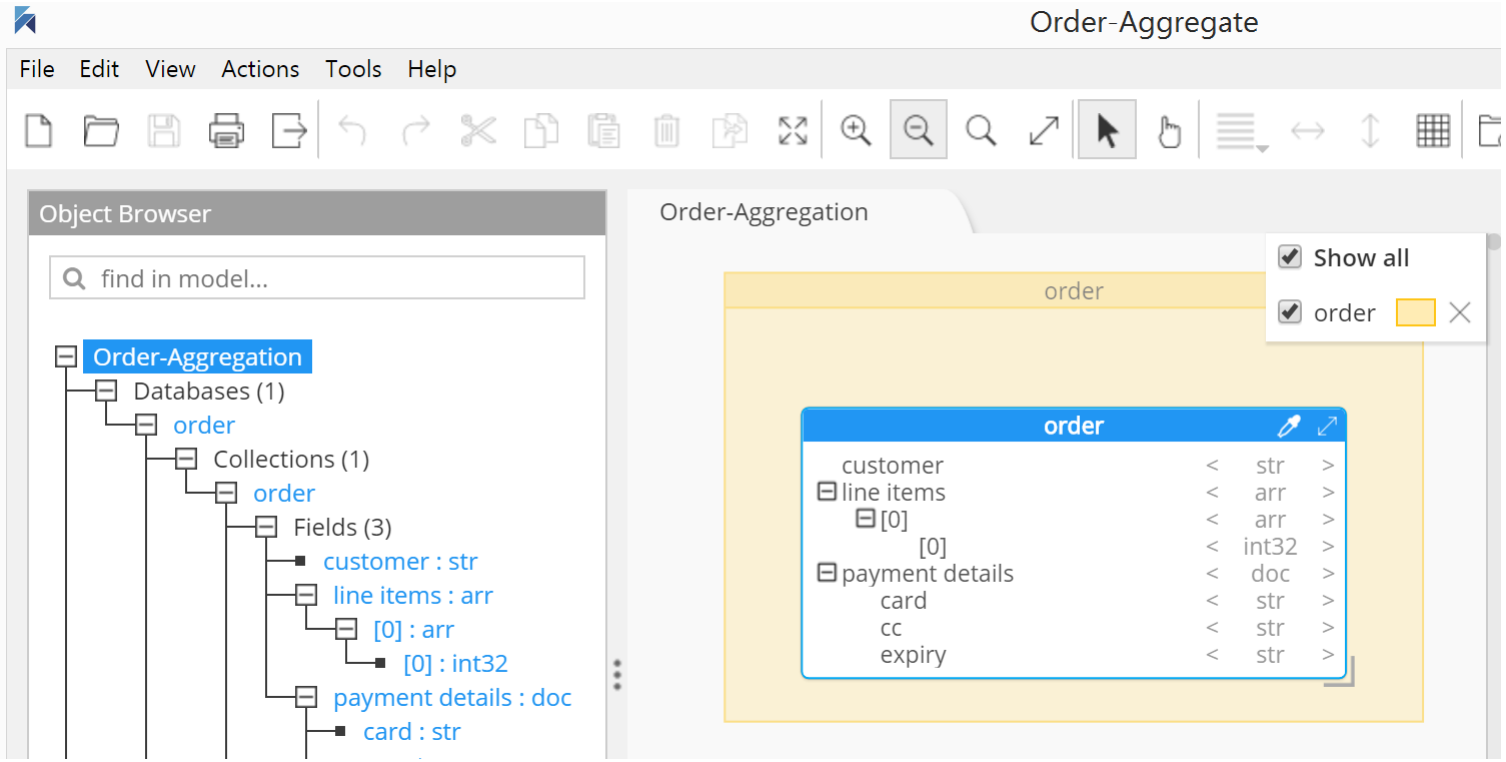
Multi Data Store Tools

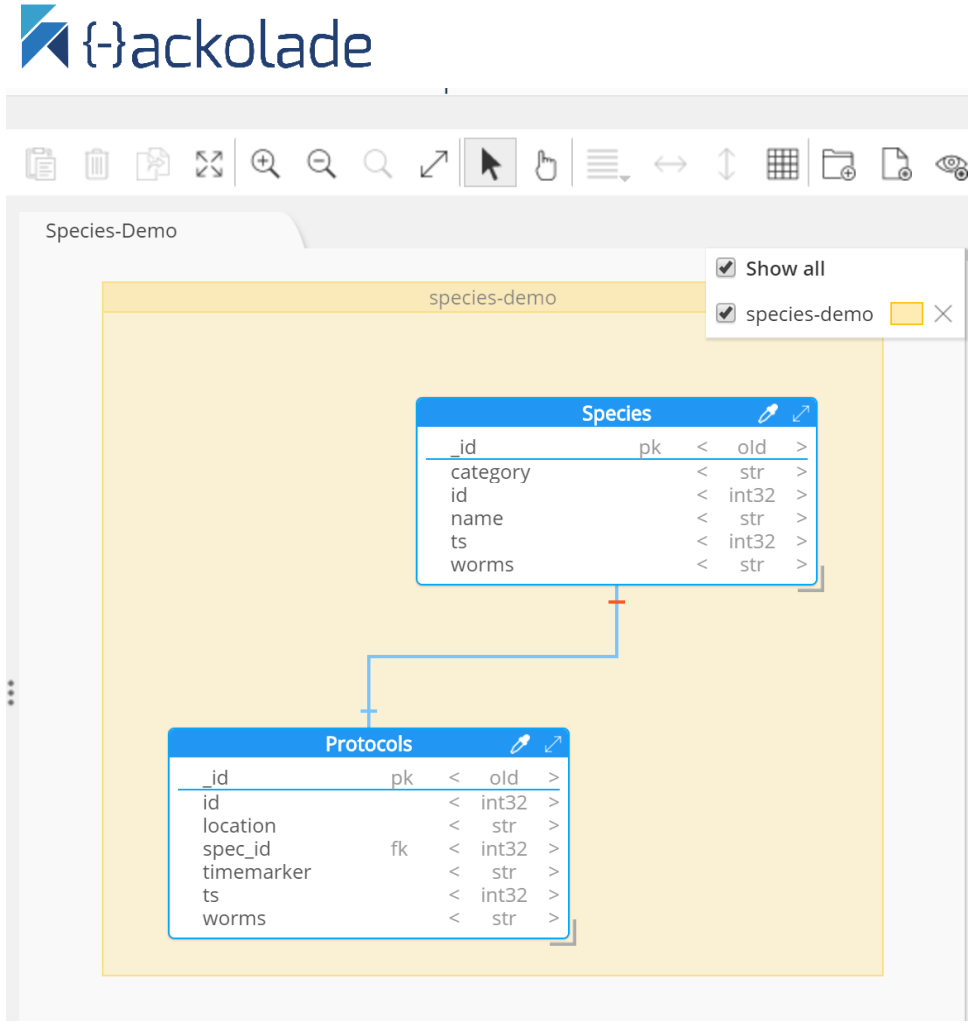
{-}ackolade erwin the data governance company

Single Data Store Tools

Studio 3T mongoDB

Forward Engineering: Schema Creation

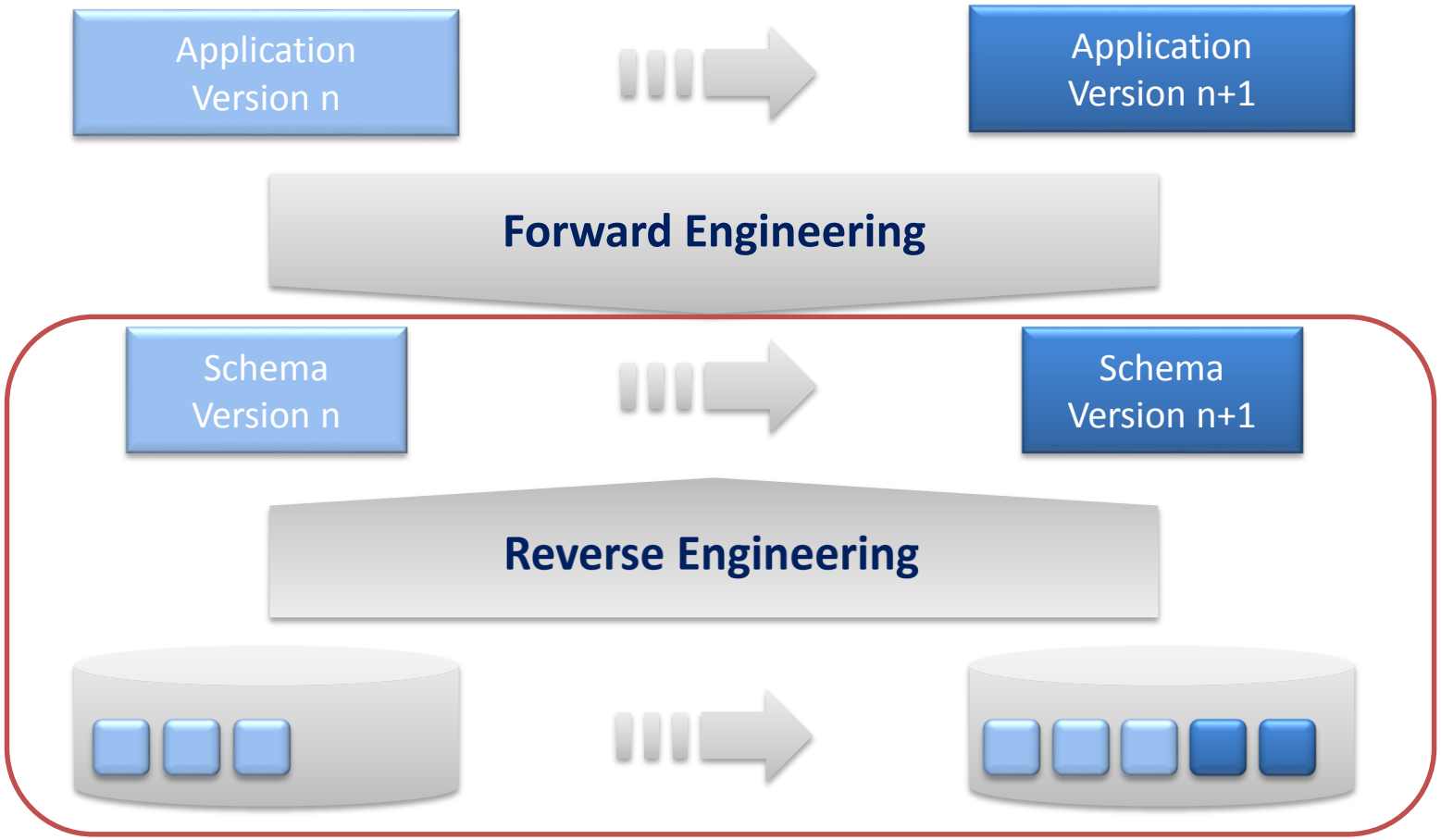




Create

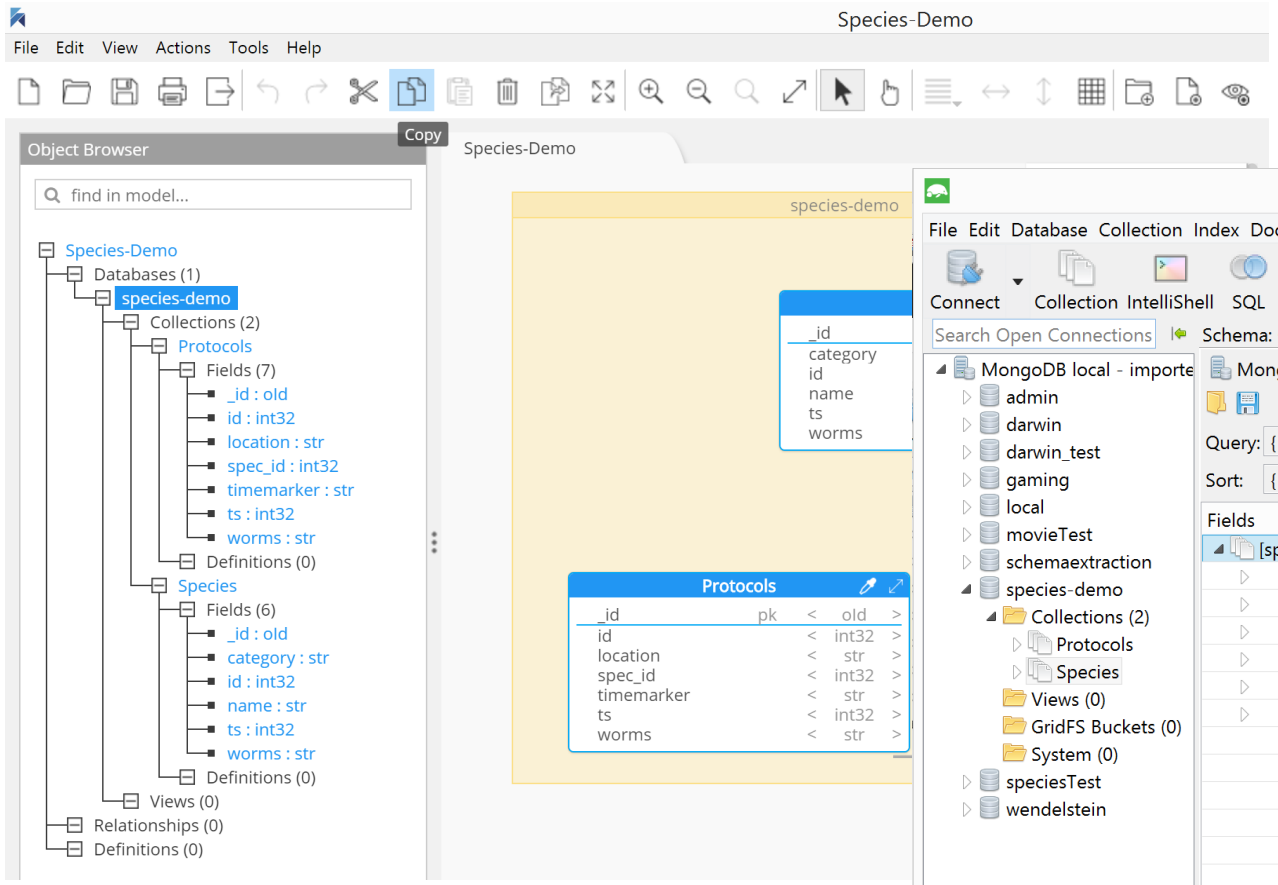
- **JSON Schema**
- Proprietary schema formats (e.g. Mongoose for MongoDB, Ottoman for Couchbase, …)

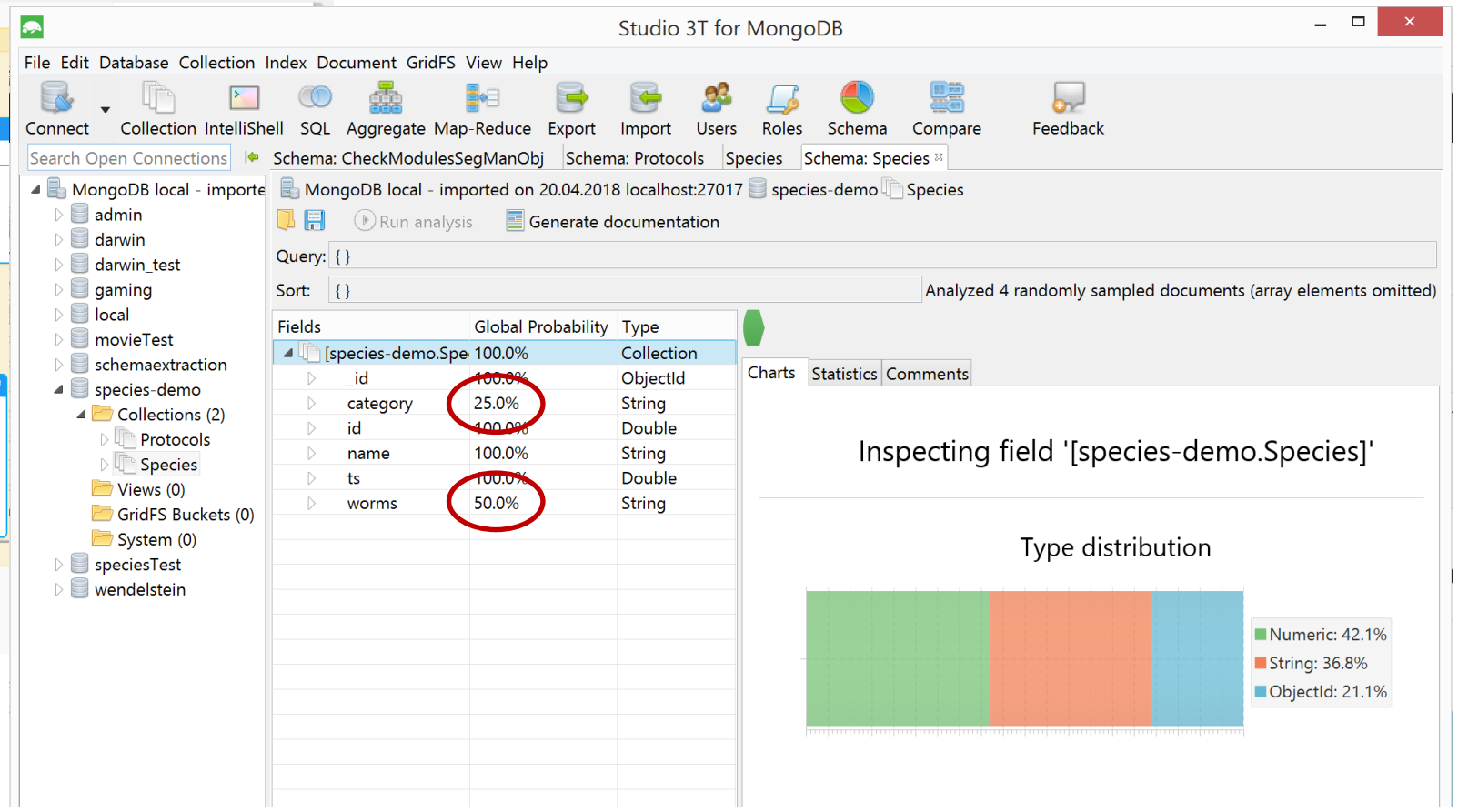
Schema Management for NoSQL Databases



- **Forward Engineering**
  - Schema Creation
- **Reverse Engineering**
  - Schema Overview
  - Data Exploration

Reverse Engineering:
Schema Overview and Data Exploration





NoSQL Schema Management: So Far, So Good

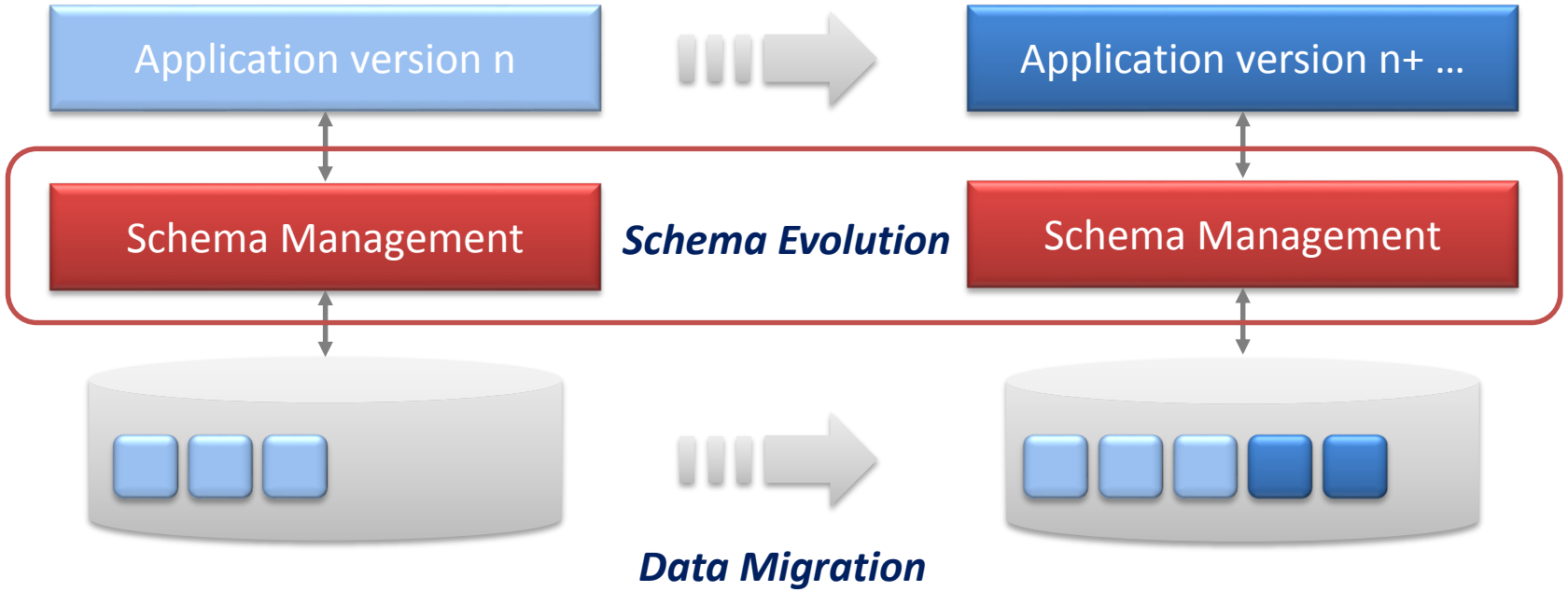
- We are able to
  - define schemas and validate data (Forward Engineering) and/or
  - extract a schema overview and explore data (Reverse Engineering)

- However, what about the **heterogeneous data** (due to different application releases, for example) in the NoSQL database?

???

Continuous Database Evolution

- (Optional) **schema management for NoSQL databases**



⇨ **Two main tasks**

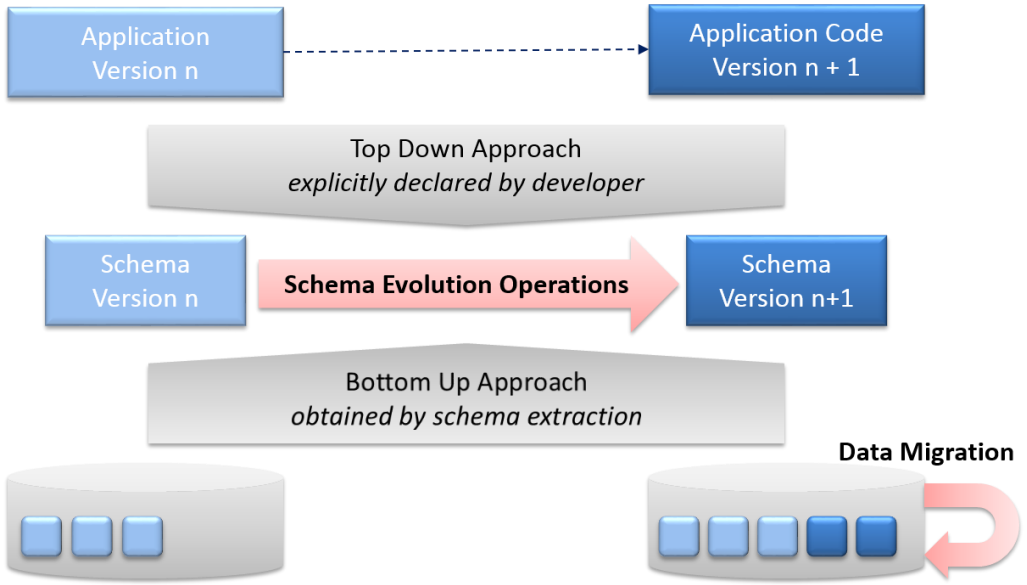
- **Schema EVOLUTION management**
- **Data migration**

NoSQL Schema Evolution Language

- Schema evolution language for describing structural changes of the data

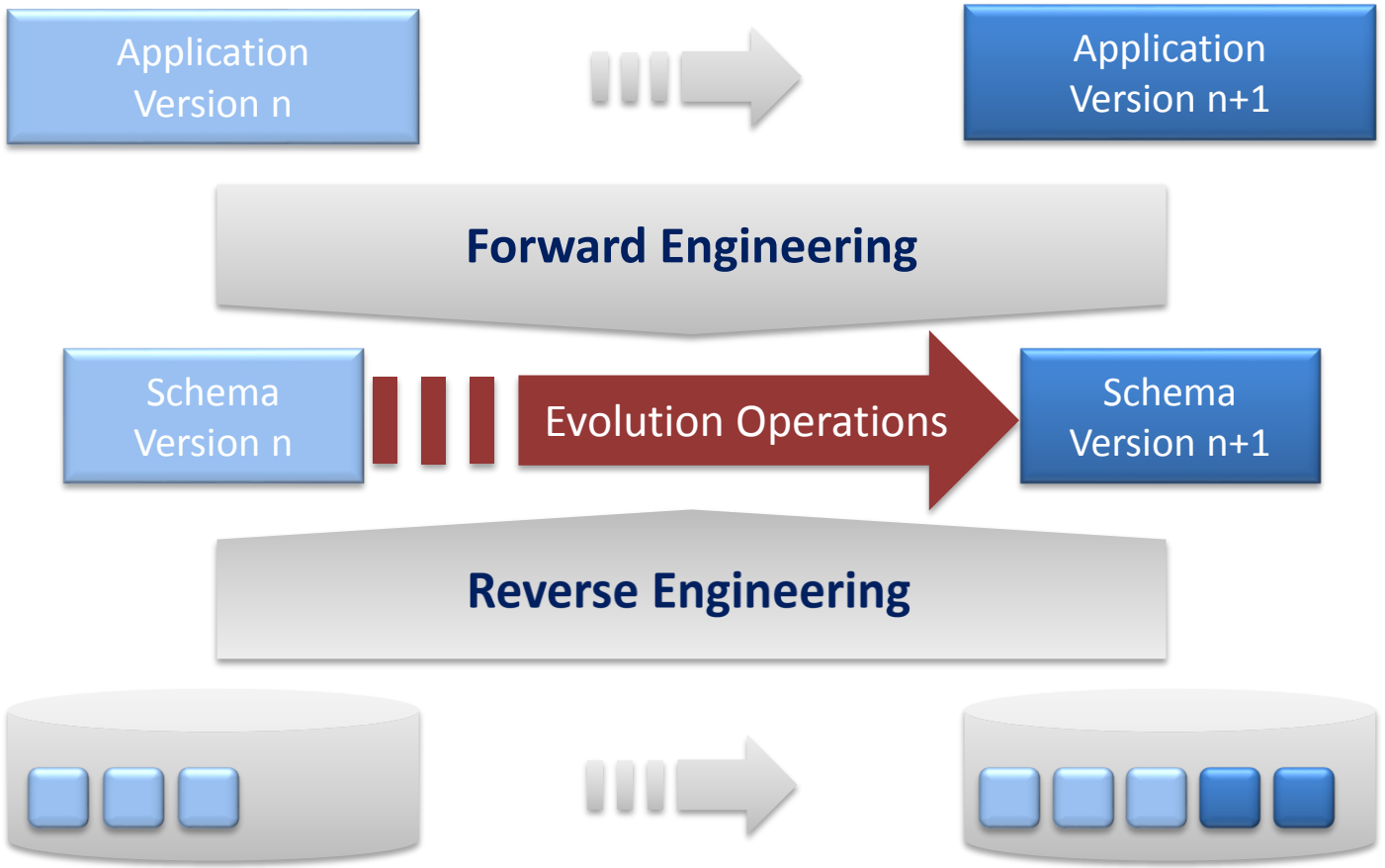
- **Single type** operations:
  - **add** property
  - **rename** property
  - **delete** property

- **Multi-type** operations:
  - **copy** property (denormalization)
  - **move** property (refactoring)



Introduced in: S. Scherzinger, M. Klettke, U. Störl: Managing Schema Evolution in NoSQL Data Store, DBPL@VLDB, 2013

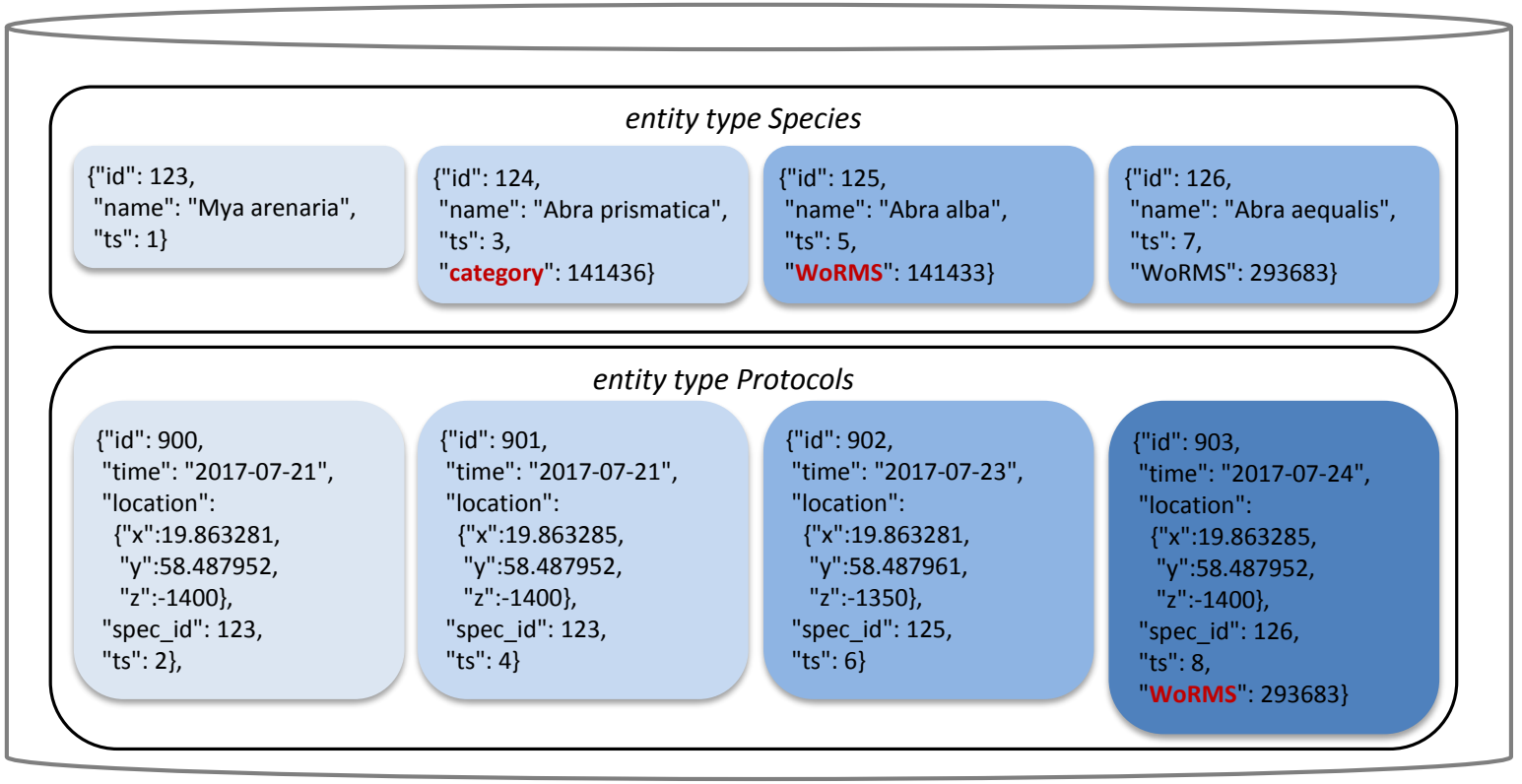
Schema Evolution Management



- **Forward Engineering**
  - Schema Creation
- **Reverse Engineering**
  - Schema Overview
  - Data Exploration
- **Advanced Reverse Engineering**
  - Schema Version Extraction

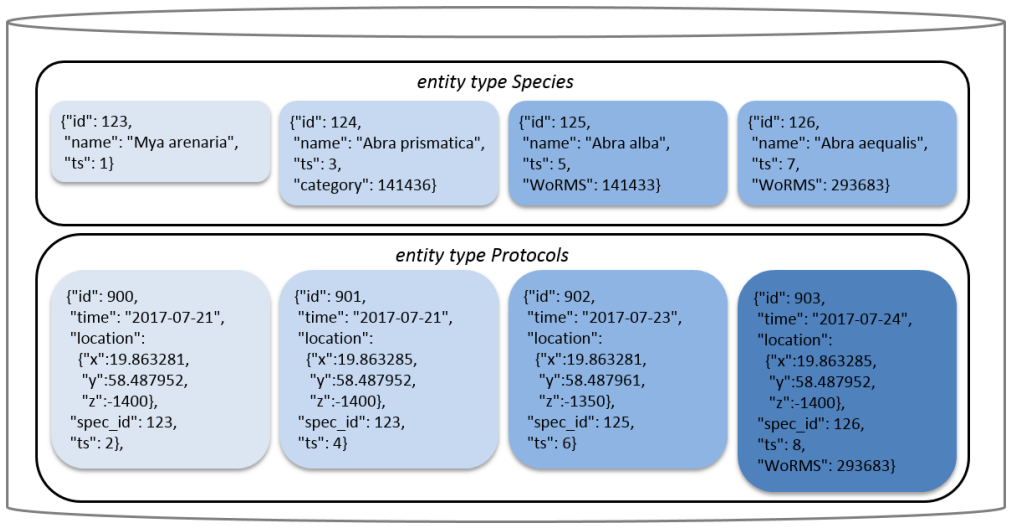
Short Excursion: Schema Version Extraction Example from Marine Biology

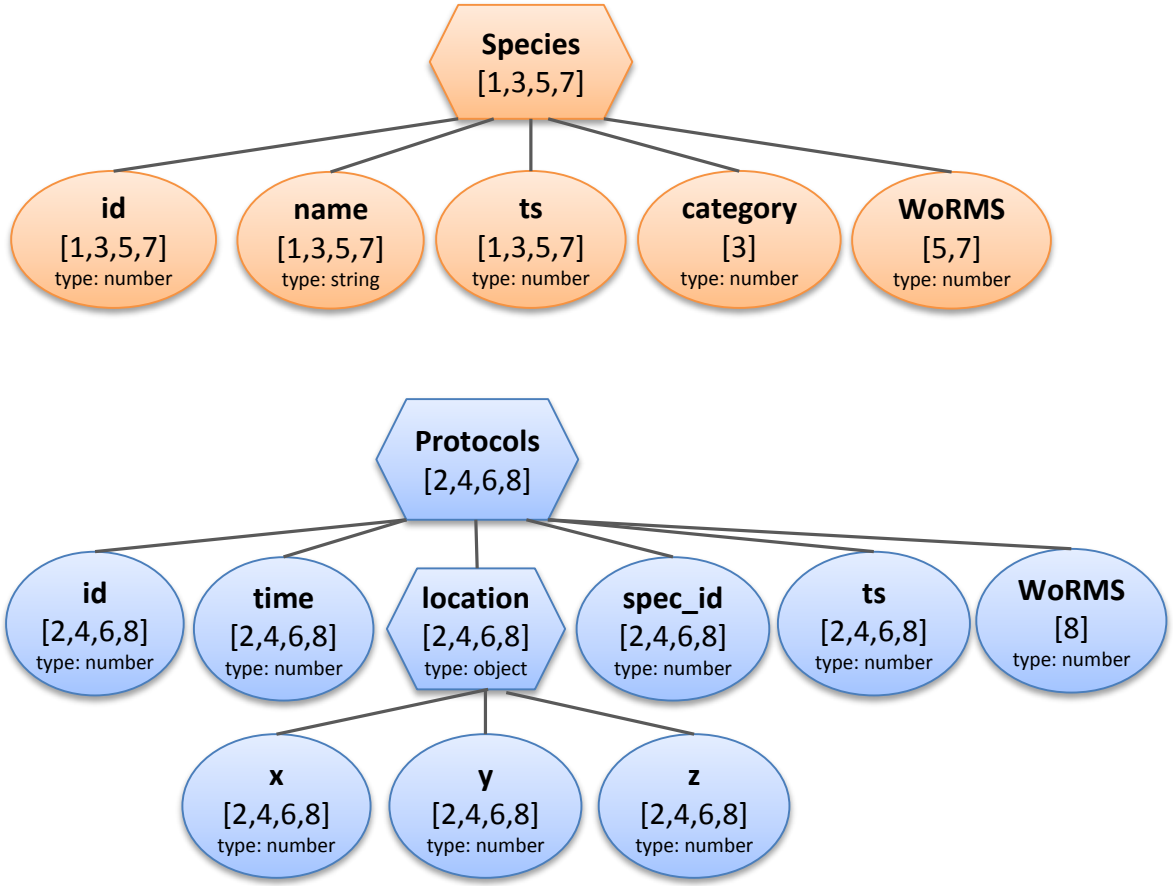
- JSON datasets for *Species* classification of the Baltic Sea and observation *Protocols*



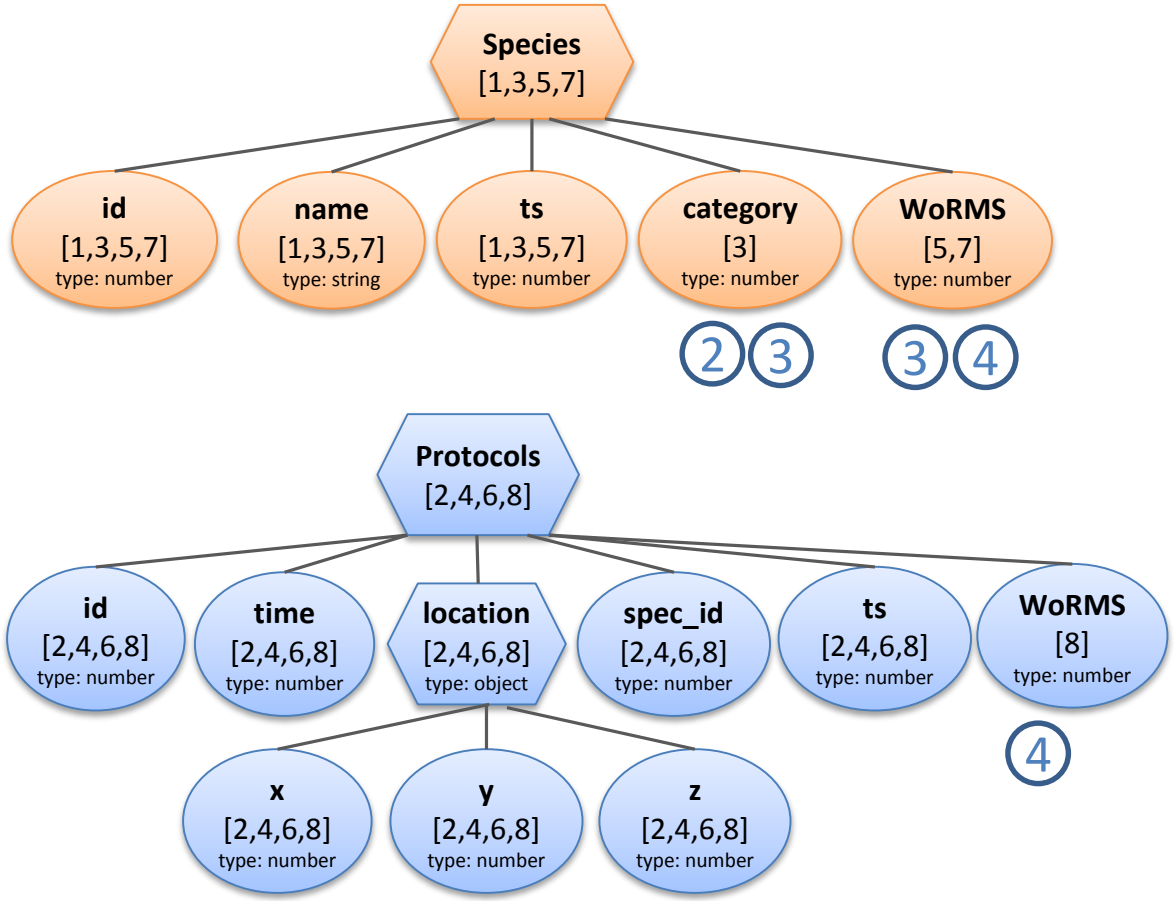
WoRMS: World Register of Marine Species

Short Excursion: Schema Version Extraction
Step 1 - Building the Schema Version Graphs





Short Excursion: Schema Version Extraction
Step 2 - Deriving Schema Evolution Operations



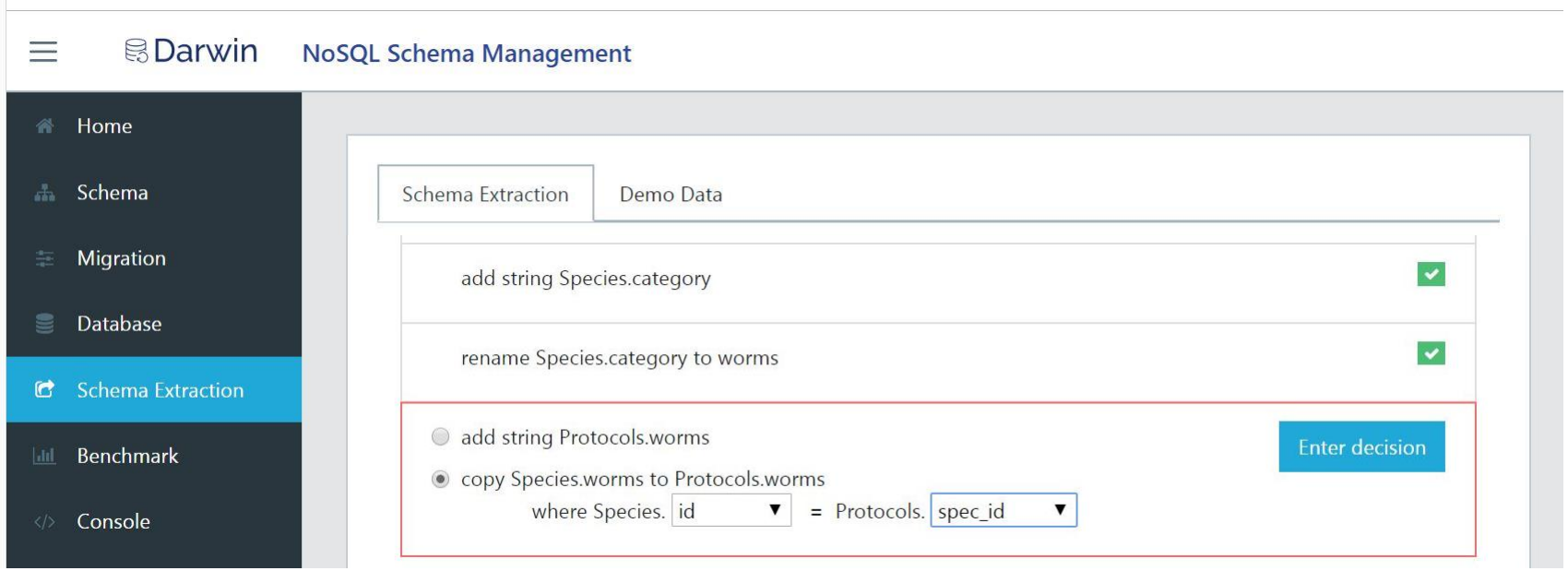
②
```
add integer Species.category
```

③
```
rename Species.category to WoRMS
```
or
```
delete Species.category
add Species.WoRMS
```

④
```
add integer Protocols.WoRMS
```
or
```
copy Species.WoRMS to Protocols.WoRMS
where Species.id = Protocols.spec_id
```

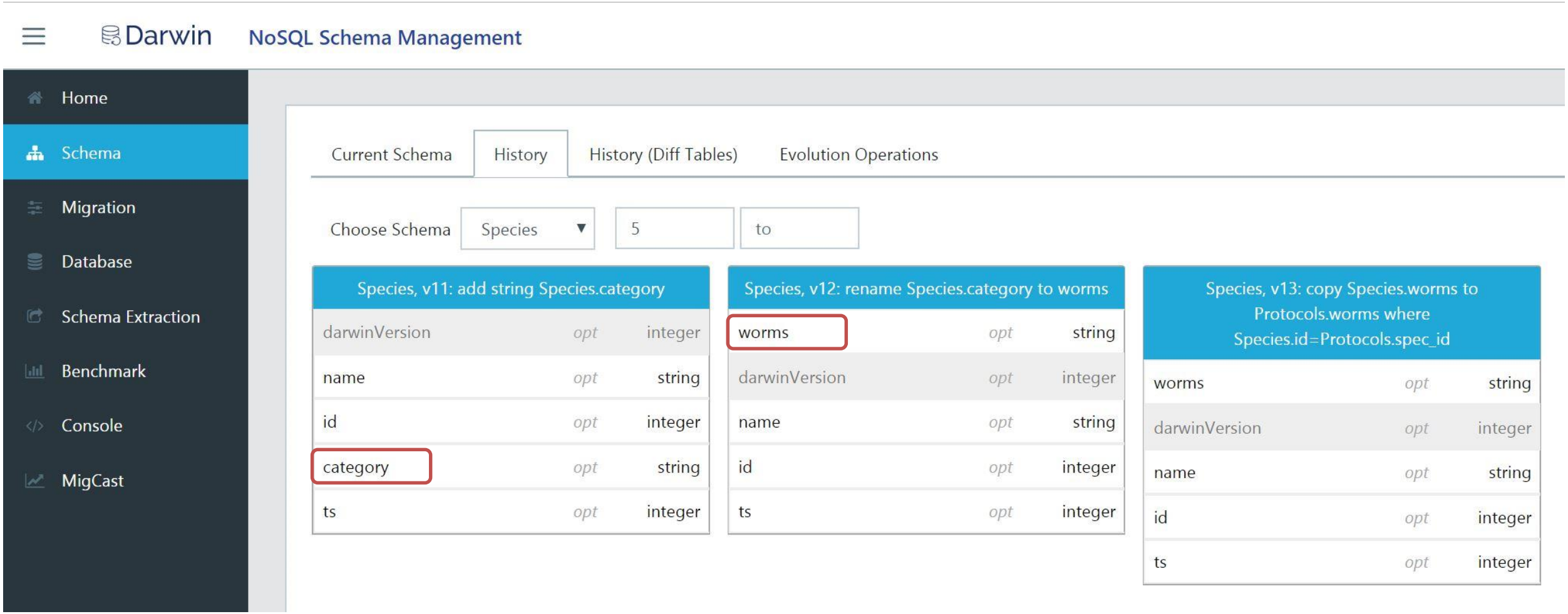
Short Excursion: Schema Version Extraction Step 3 - Resolving Ambiguities

- **Interactively resolving ambiguous schema evolution operations:**
  - Alternative schema evolution operations
  - Specifying join conditions for move or copy operations



- **Next steps**
  - Automated choice in case of ambiguities
  - Suggestion of meaningful join conditions
  - **Approach to a solution:** Algorithm for deriving inclusion dependencies from NoSQL datasets proposed in M. Klettke, H. Awolin, U. Störl, D. Müller, and S. Scherzinger. Uncovering the Evolution History of Data Lakes. SCDM@BigData 2017

Advanced Reverse Engineering: Schema Version Extraction

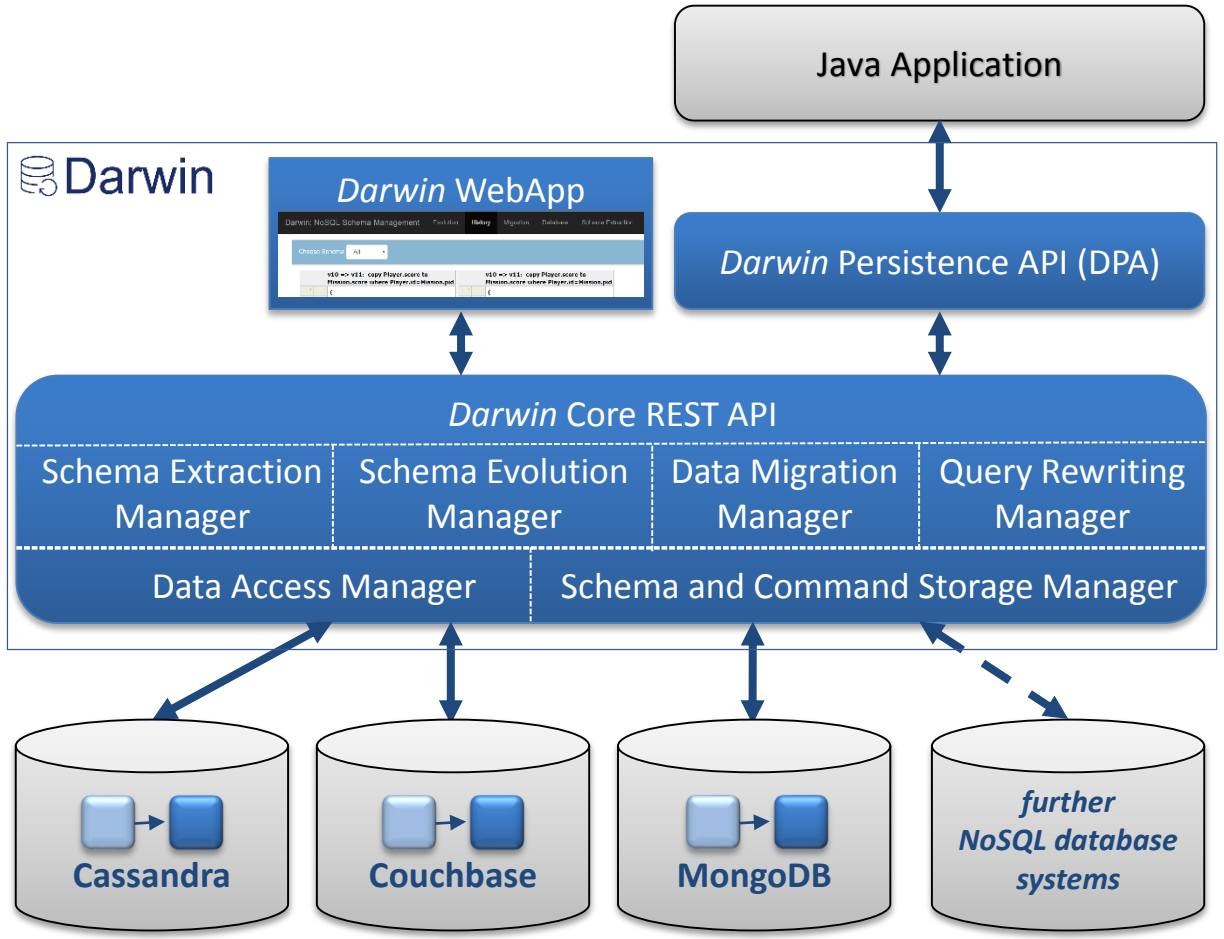


Our Approach: Supporting Schema Management and Data Migration with *Darwin*

***Darwin* supports the complete schema management life cycle:**

- declaring or extracting an initial schema,
- repeatedly applying schema changes,
- proposing mappings between legacy schema versions,
- and migrating legacy data.

The system is implemented for different types of NoSQL database systems.



U. Störl, D. Müller, J. Stenzel, A. Tekleab, S. Tolale, M. Klettke, S. Scherzinger. Curating Variational Data in Application Development. ICDE 2018.

NoSQL Schema Management – Tools (Selection)

	Multi Data Store Tools	Single Data Store Tools	Research Prototypes
• **Forward Engineering** – Schema Creation			
• **Reverse Engineering** – Schema Overview – Data Exploration		 	 *NoSQL DEP*
• **Advanced Reverse Eng.** – Schema Version Extraction			 *NoSQL DEP*

Darwin: Darmstadt University of Applied Sciences, University of Rostock, OTH Regensburg, Germany: https://fbi.h-da.de/personen/uta-stoerl/dfg-projekt-nosql-schema-evolution/
NoSQL DEP: University of Murcia, Spain: https://www.researchgate.net/project/NoSQL-Data-Engineering

Schema Evolution Management: Further Reading

- S. Scherzinger, M. Klettke, U. Störl: Managing Schema Evolution in NoSQL Data Store, DBPL@VLDB, Italy, 2013

- M. Klettke, H. Awolin, U. Störl, D. Müller, and S. Scherzinger. Uncovering the Evolution History of Data Lakes. SCDM@BigData, Washington, USA 2017

- A. H. Chillón, S. F. Morales, D. Sevilla, J. G. Molina: Exploring the Visualization of Schemas for Aggregate-Oriented NoSQL Databases. ER 2017

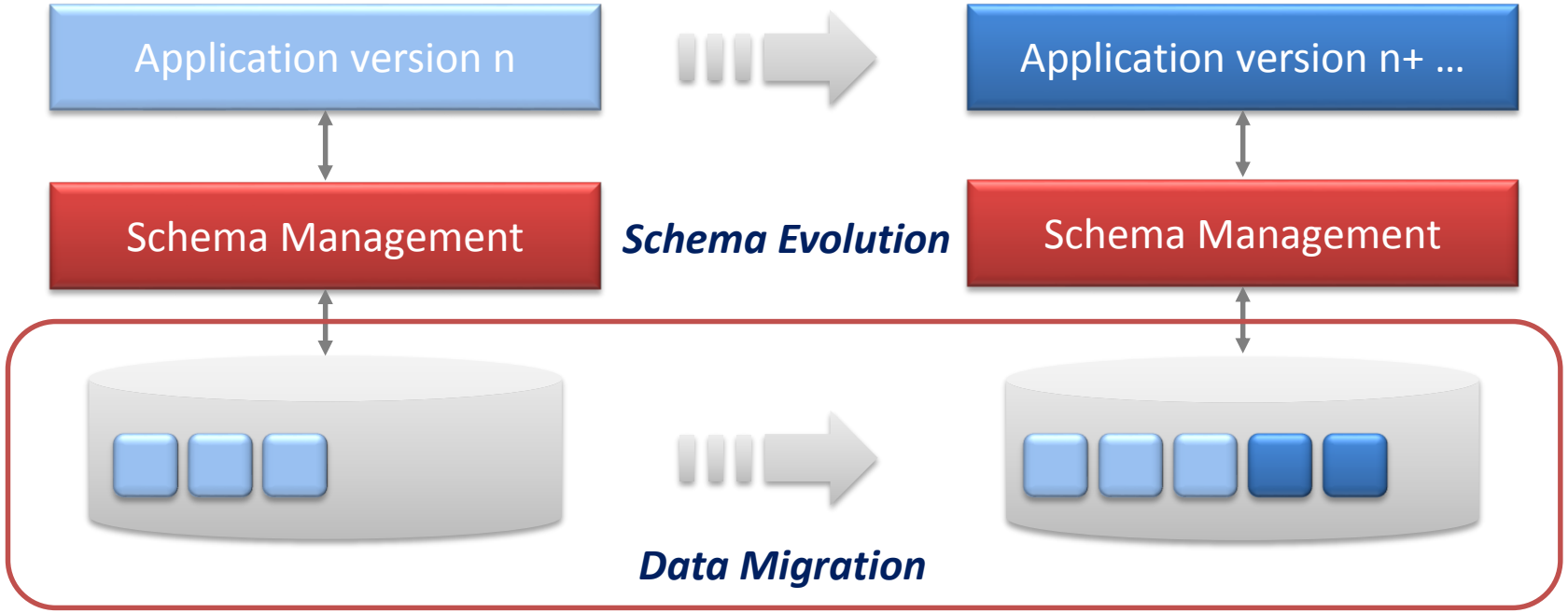
- U. Störl, D. Müller, J. Stenzel, A. Tekleab, S. Tolale, M. Klettke, S. Scherzinger. Curating Variational Data in Application Development. ICDE, Paris, 2018

Agenda

- Motivation
- NoSQL Schema Evolution Management
- Big Data Migration in the Cloud

Continuous Database Evolution

- (Optional) **schema management for NoSQL databases**



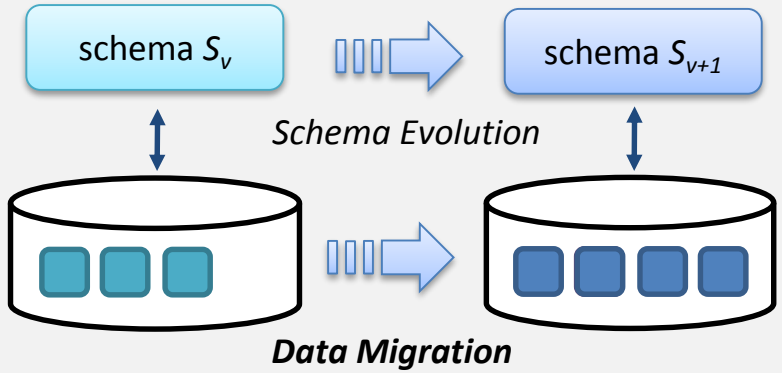
⇨ **Two main tasks**

- **Schema evolution management**
- **Data migration** as safe process based on schema evolution management

Basic Strategies of Data Migration

Eager Migration

- after introduction of *a new schema version*, all entities are migrated



Advantages:

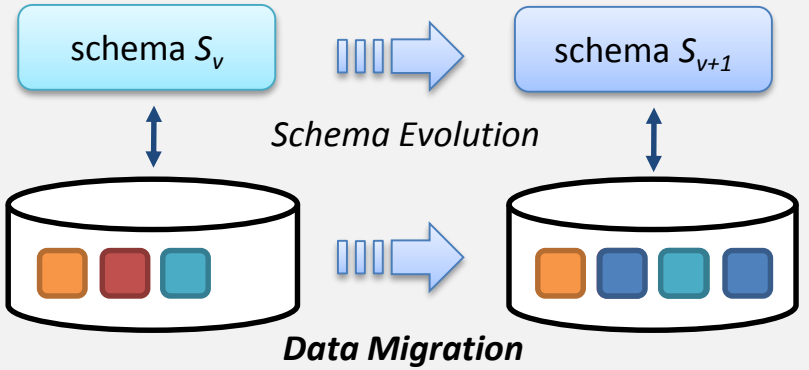
+ all entities are in the current version
+ **low latency** (when entities are accessed)

Disadvantages:

– even entities that are not in use are migrated
– **high number (and costs) of migration operations**

Lazy Migration

- evolution operations are stored,
- data migration is done on request



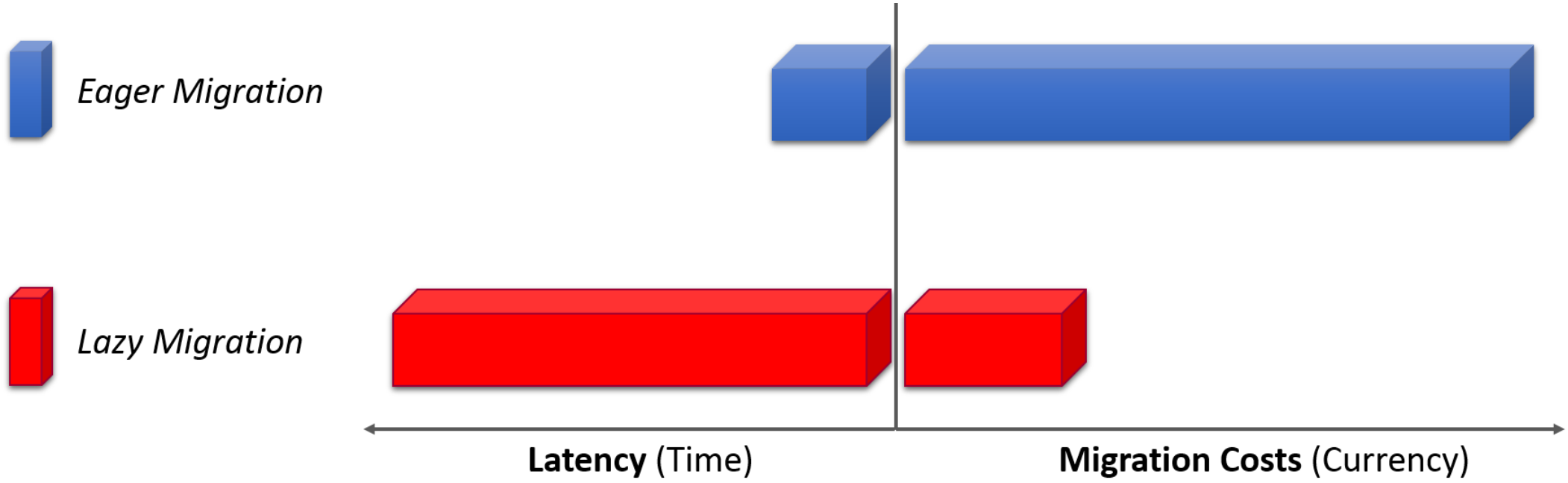
Advantages:

+ only entities that are in use are migrated
+ **no unnecessary** data migration **operations**
+ **composition of operations** is possible

Disadvantages:

– entities in the NoSQL database in different versions
– increased **latency**

Tradeoffs in Choosing a Data Migration Strategy



How to Reduce Costs of Data Migration?

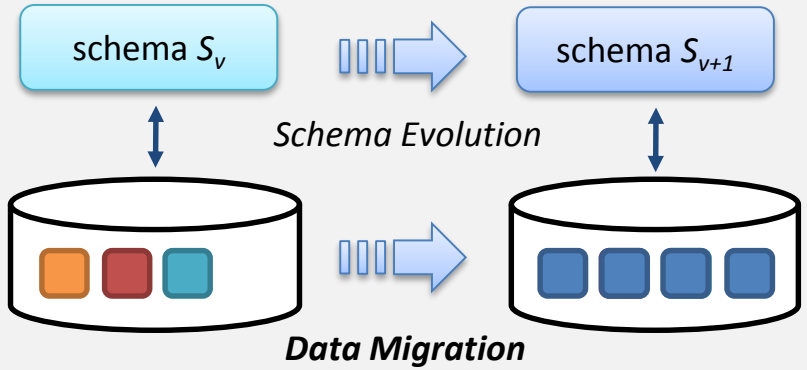
- In case of **large amount of datasets**
  - update operations even for cold data (that not in use)
- In case of **database as a service**
  - payable operations, monetary costs for all data migrations

- **How to reduce costs of data migration?**
  - ⇨**Optimize Data Migration**
  - ⇨**Hybrid / Proactive Migration Approaches**
    - **Predictive Migration**
    - Incremental Migration

Proactive Migration Strategies

Incremental Migration

- in some versions, an eager migration is applied



Advantages:

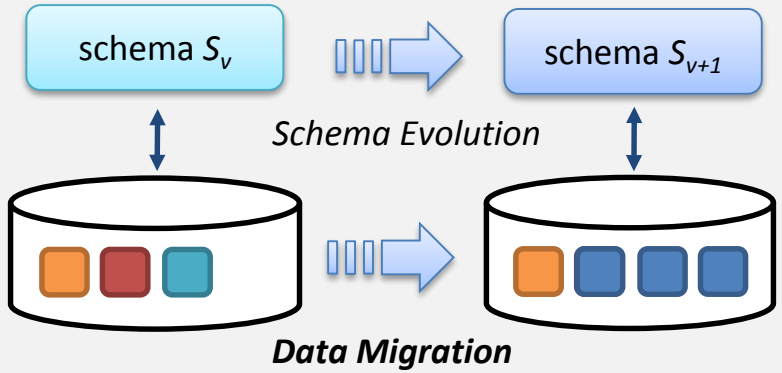
+ **lower migration costs**
+ **composition of operations** is possible

Disadvantage:

– even entities that are not in use are migrated

Predictive Migration

- forecast function which entities are accessed in near future (based on heuristics)
- predictive migration of these entities



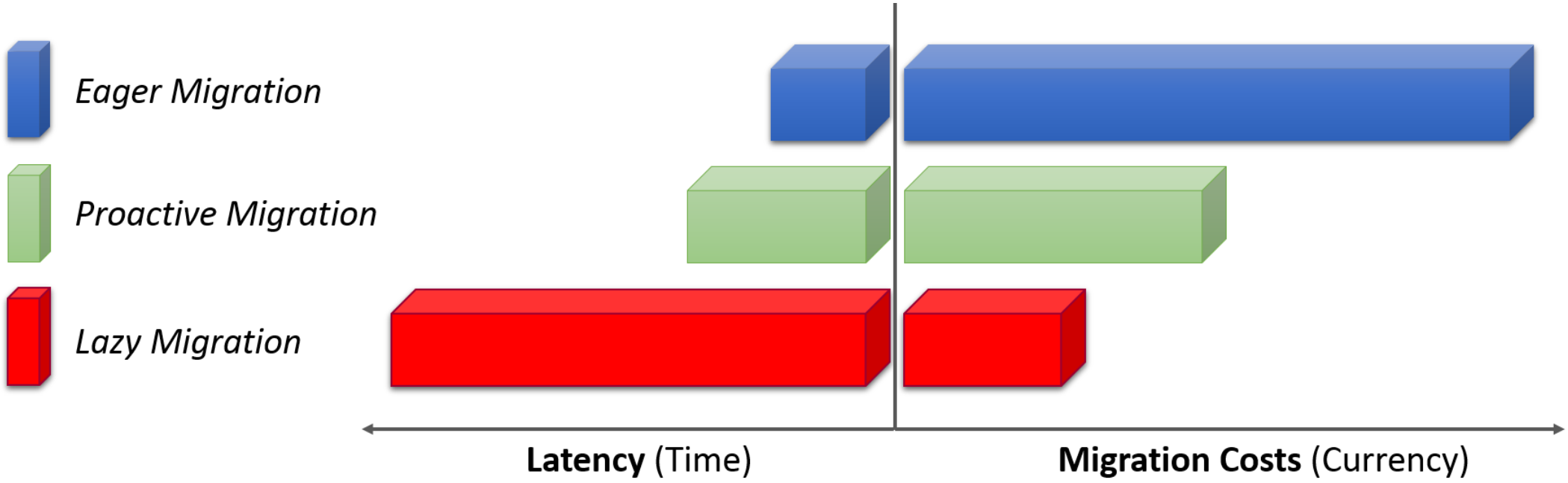
Advantages:

+ **decreased average latency**
+ **reduced** number of migration **operations (and costs)**

Disadvantage:

– additional migration operations in case of wrong predictions

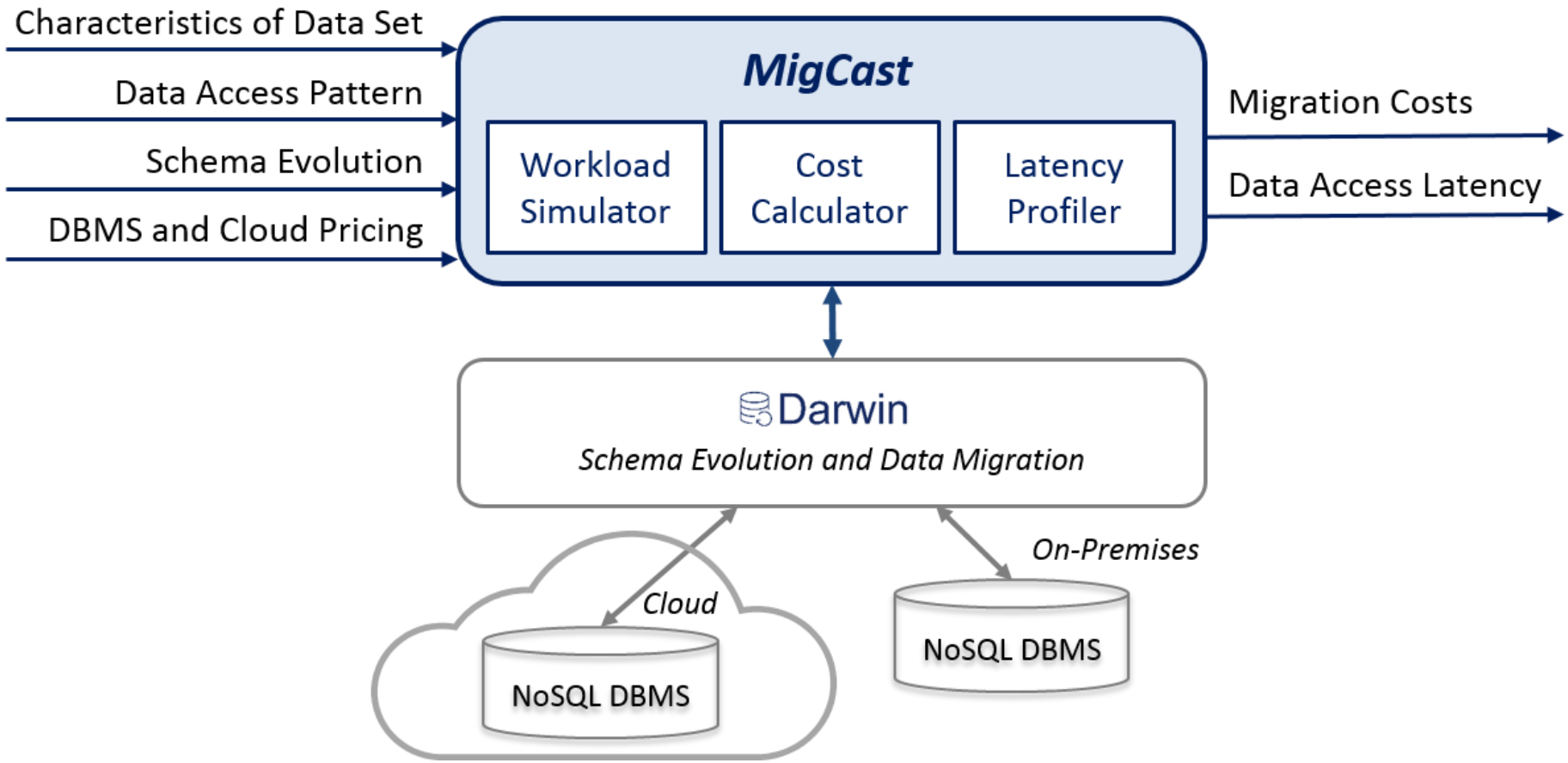
Tradeoffs in Choosing a Data Migration Strategy



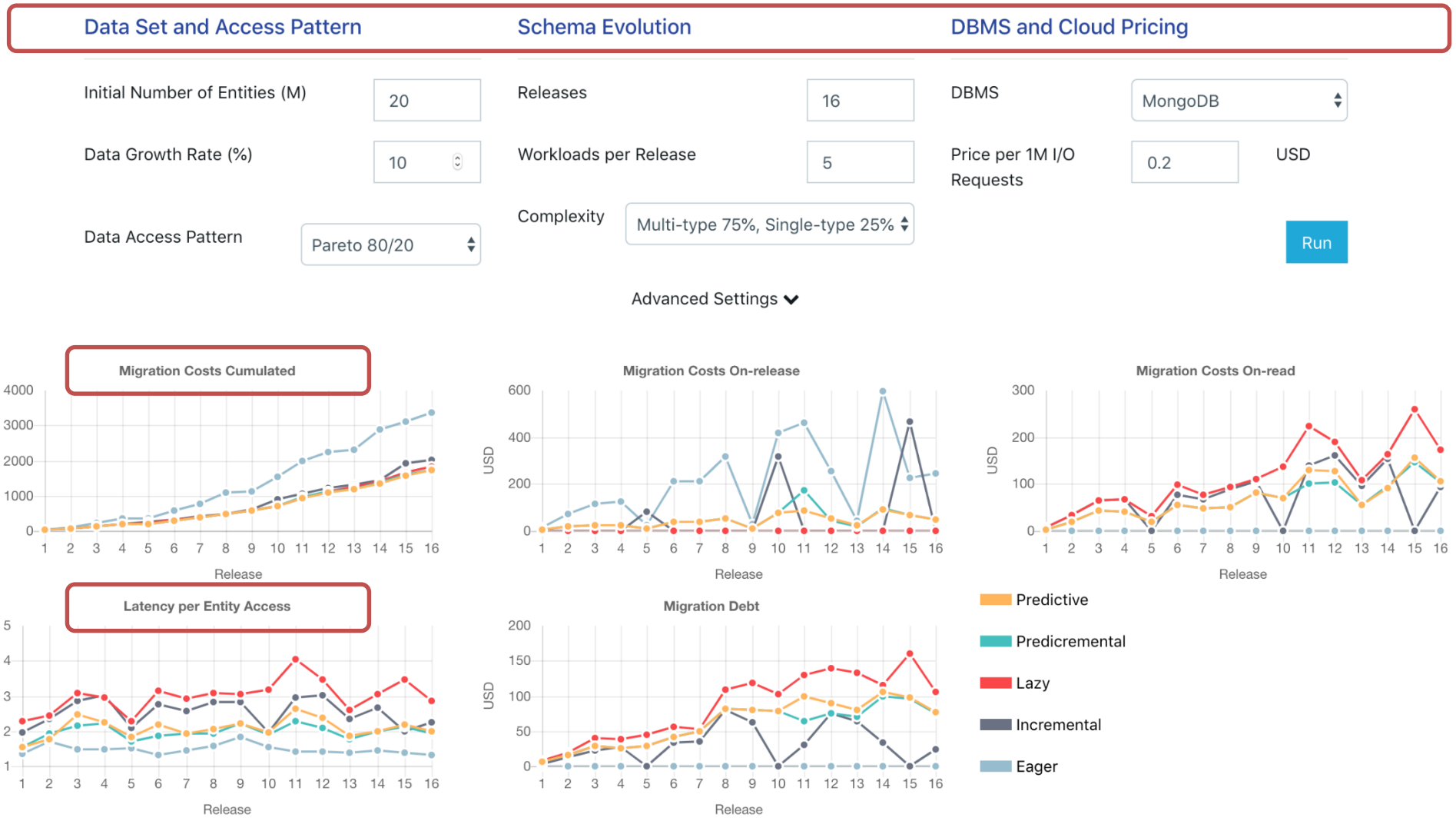
Choosing the Right Data Migration Strategy

*U. Störl, A. Tekleab, M. Klettke, S. Scherzinger: In for a Surprise When Migrating NoSQL Data. ICDE 2018.

MigCast: Choosing a Suitable Data Migration Strategy



MigCast: Choosing a Suitable Data Migration Strategy



A. Hillenbrand, M. Levchenko, U. Störl, St. Scherzinger, M. Klettke
MigCast: Putting a Price Tag on Data Model Evolution in NoSQL Data Stores, **Demo@SIGMOD 2019**

Darwin

This work is supported by DFG 385808805

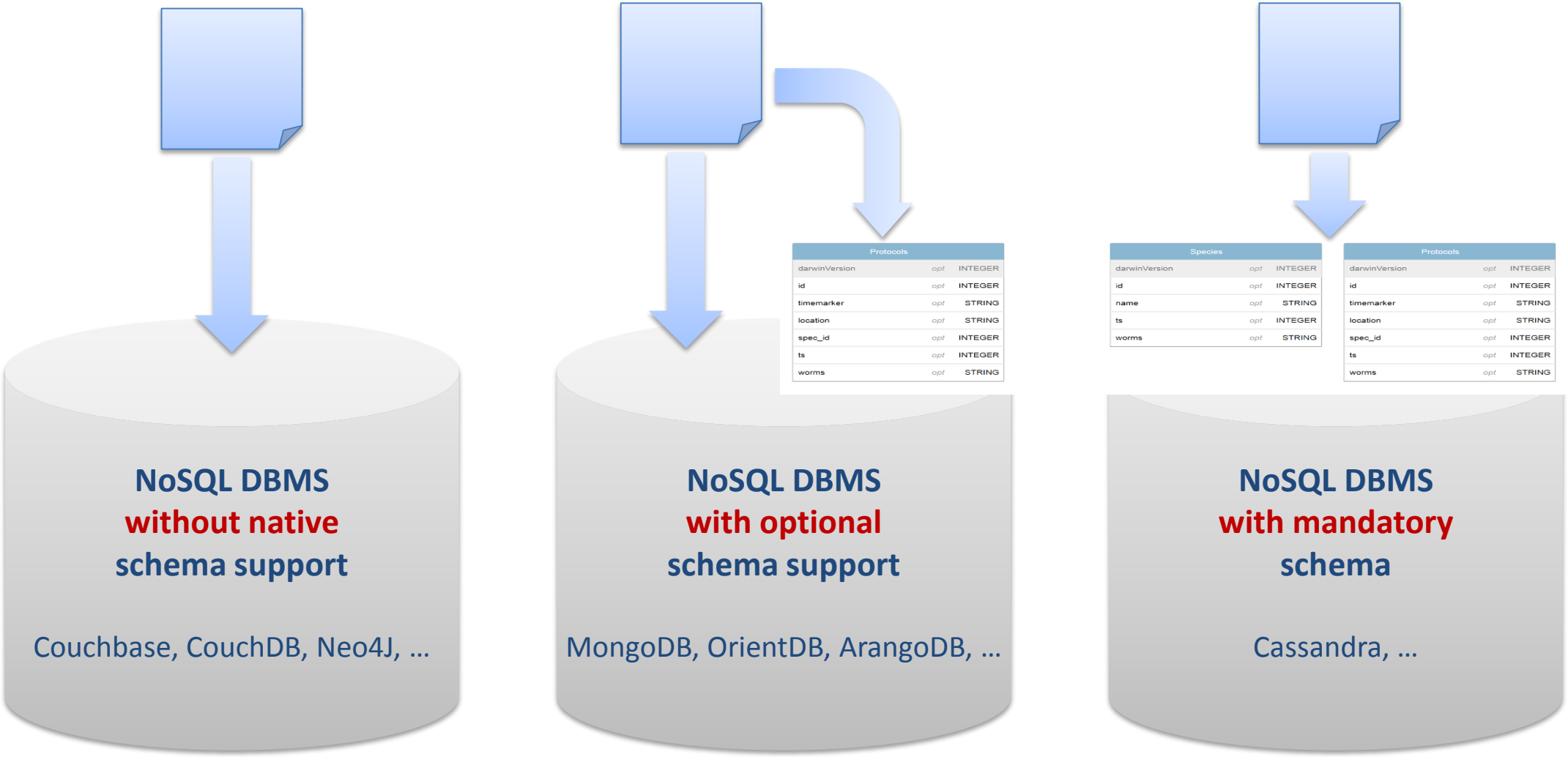
Conclusion and Outlook

- **Schema Evolution Management and**
- **Big Data Migration**
- **Further Research work**
  - **Evolution Management in Multi-Model Databases and PolyStores**
  - **Craft a dedicated NoSQL schema evolution and data migration benchmark**
  - Cost function to support a decision about the appropriate data migration strategy and its parametrization

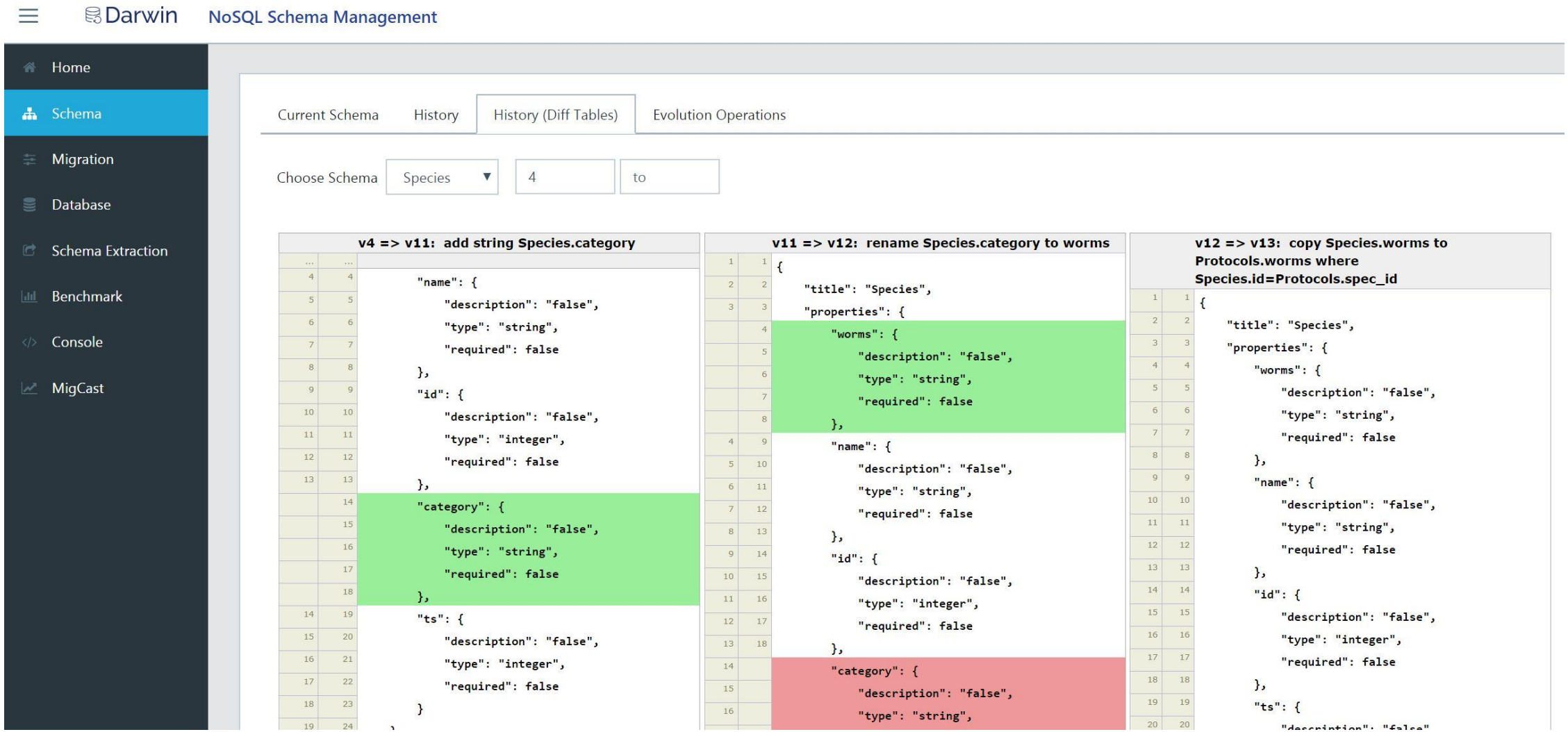
  ⇨ **Design and develop a data migration advisor**
- **DFG-Project “NoSQL Schema Evolution and Big Data Migration at Scale”**
  - https://fbi.h-da.de/personen/uta-stoerl/dfg-projekt-nosql-schema-evolution/

Backup

Remark: NoSQL Database Are Schema-Free – Aren't They?



Advanced Reverse Engineering: Schema Version Extraction



Approaches to Realize Data Migration

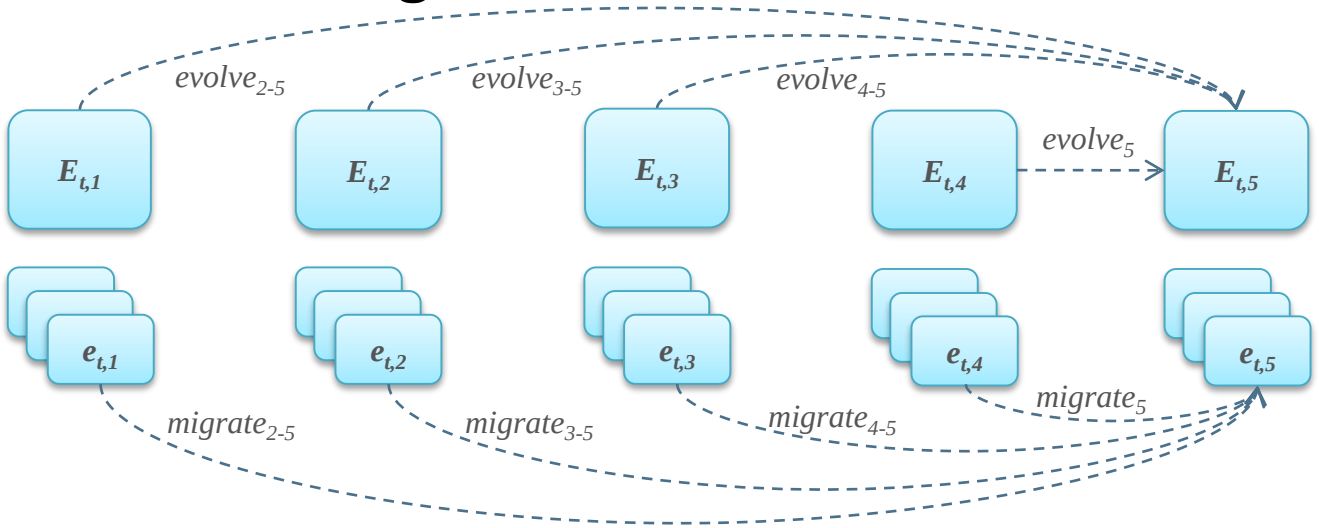
- **Custom-coded Migration Scripts**
  - Error-prone and expensive ☹
- **Using Object-NoSQL-Mapper Annotations**
  - Easy to realize for simple evolution operations like add, delete, and rename (e.g. @AlsoLoad)
  - More expensive for complex operations like split, merge, copy, and move (coding @PostLoad methods)

```
@Entity
public class Profile {
    @Id
    int profileID;
    String firstname;
    String lastname;

    @AlsoLoad("year")
    int yearOfBirth;
    String country;
}
```

Optimization of Data Migration

- In case, entities are migrated from older versions:



- **Composition of evolution operations** is possible in some cases:
  - ⇨smaller number of data migration operations
  - ⇨lower migration costs

Introduced in: M. Klettke, U. Störl, M. Shenavai, S. Scherzinger: NoSQL Schema Evolution and Big Data Migration at Scale, 4th Scalable Cloud Data Management Workshop (SCDM) @ IEEE Big Data Conference, Washington, D.C., 2016

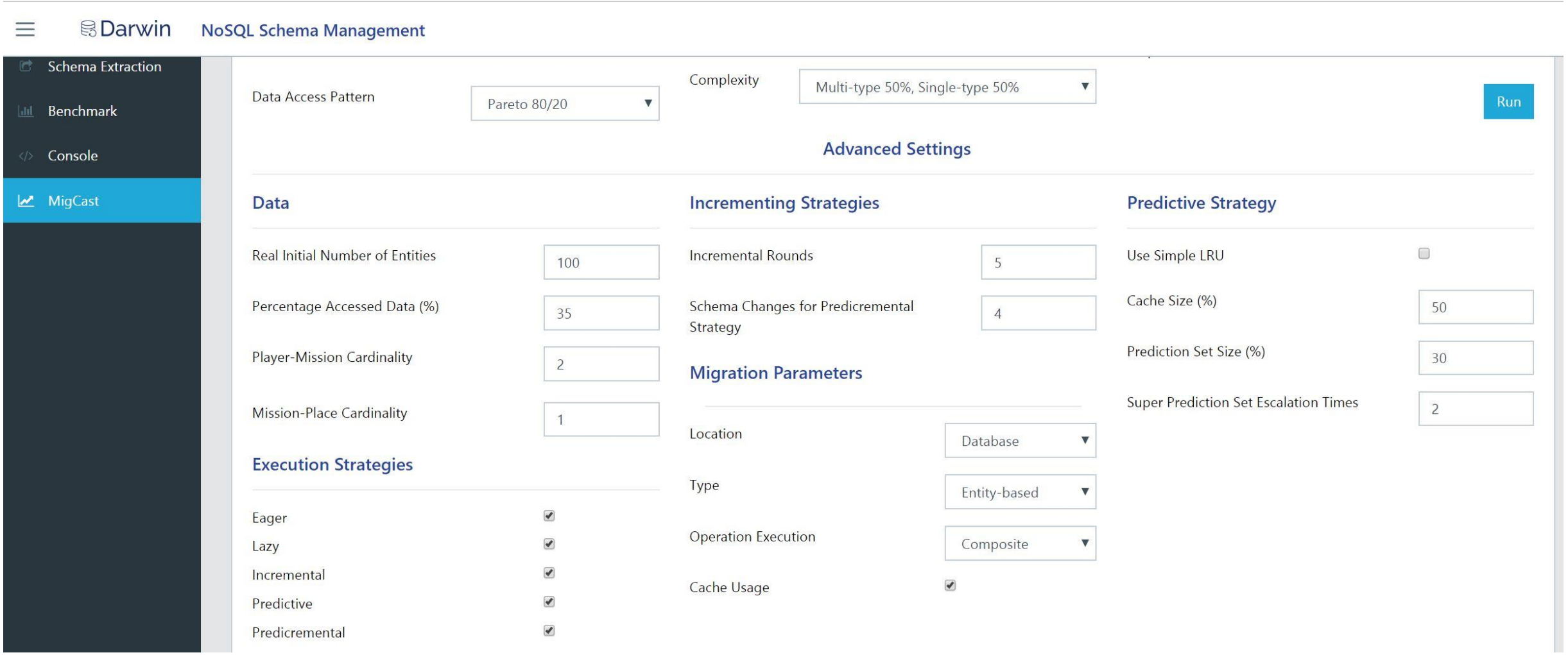
Optimization of Data Migration

- Composition rules for two evolution operations, first *op*1 and then *op*2

*op*1 \ *op*2	**rename** $E_B.y$ **to** z	**copy** $E_B.y$ **to** $E_C.z$ $cond_2$	**move** $E_B.y$ **to** $E_C.z$ $cond_2$	**delete** $E_B.y$
add $E_B.y$ = *default*	**add** $E_B.z$ = *default*	-	**add** $E_C.z$ = *default* $cond_2$ **add** $E_B.y$ = *default* $\neg cond_2$	ϵ (noop)
rename $E_B.x$ **to** y	**rename** $E_B.x$ **to** z	**copy** $E_B.x$ **to** $E_C.z$ $cond_2$	**move** $E_B.x$ **to** $E_C.z$ $cond_2$ **rename** $E_B.x$ **to** y $\neg cond_2$	**delete** $E_B.x$
copy $E_A.x$ **to** $E_B.y$ $cond_1$	**copy** $E_A.x$ **to** $E_B.z$ $cond_1$	-	**copy** $E_A.x$ **to** $E_C.z$ $cond_1 \wedge cond_2$ **copy** $E_A.x$ **to** $E_B.y$ $cond_1 \wedge \neg cond_2$	ϵ (noop)
copy $E_B.y$ **to** $E_D.u$ $cond_3$	-	-	-	**move** $E_B.y$ **to** $E_D.u$ $cond_3$
move $E_A.x$ **to** $E_B.y$ $cond_1$	**move** $E_A.x$ **to** $E_B.z$ $cond_1$	-	**move** $E_A.x$ **to** $E_C.z$ $cond_1 \wedge cond_2$ **move** $E_A.x$ **to** $E_B.y$ $cond_1 \wedge \neg cond_2$	**delete** $E_A.y$

Introduced in: M. Klettke, U. Störl, M. Shenavai, S. Scherzinger: NoSQL Schema Evolution and Big Data Migration at Scale, 4th Scalable Cloud Data Management Workshop (SCDM) @ IEEE Big Data Conference, Washington, D.C., 2016

MigCast: Advanced Settings



Supporting Schema Management and Data Migration with *Darwin*: Measurement Environment

- **Big Data Cluster @ FBI**
  - 40 Nodes
    - 64 GB / 128 GB RAM; 4 TB storage each
  - MongoDB / Couchbase / Cassandra / …
  - Apache Hadoop
  - Apache Spark
  - Big Data Competence Centre: http://fbi.h-da.de/bdcc