Management of Heterogeneous Data in Traditional and non Traditional Databases

Paolo Atzeni

ROMA TRE
UNIVERSITÀ DEGLI STUDI

Based on recent work with F. Bugiotti, L. Rossi
and on work done over many years with R. Torlone, L. Bellomarini, P.A. Bernstein, P. Cappellari, G. Gianforme

Taormina, May 2012

Heterogeneity … of mountains

Glaciers …

…rocks …

… volcanos

Yesterday on the shuttle bus

- We were discussing about languages and dialects
- A colleague said:
  - Sanscrit is "more general than any other language, it has twelve case …"

Outline

- Heterogeneity in interoperabilty and migration settings
- Management of multiple models with a metamodel approach
- Heterogeneity in NoSQL systems
- A common interface for NoSQL systems
- Future work

Heterogeneity

- Despite all standardization efforts, many data models exist
- With different features and goals
  - semantic models and logical models:
    - E-R, functional, (conceptual) object
    - relational, network, object
  - general purpose models as well as problem oriented models (for specific contexts: DW, statistical, spatial, temporal)
  - more models recently with the Web and XML
  - Yet more with NoSQL
- Variations of models
  - versions within a family:
    - many versions of the ER model, of the OR, many NoSQL

When and how we handle heterogeneity

- In design of complex heterogeneous systems
  - the results of independent design activities need to be integrated or exchanged
- In transition, migration, consolidation, ETL and similar settings
  - there is the need to transform data in an "off-line" way
- In heterogeneous operational systems
  - interoperability at run-time is needed

Outline

- ✓ Heterogeneity in interoperabilty and migration settings
- MIDST: heterogeneity and translation with traditional models
- Heterogeneity in NoSQL systems
- A common interface for NoSQL systems
- Future work

MIDST
(Model Independent Schema and Data Translation)

- Schema and data translation
  - initially with an off-line approach
  - later also with a run-time one
- Model-generic:
  - works for many models, in an extensible way
- Model-aware:
  - models are described

Off-line schema and data translation

- Schema translation:
  - given
    - schema S1 in model M1 and
    - model M2
  - find a schema S2 in M2 that “corresponds” to S1
- Schema and data translation:
  - given also a database D1 for S1
  - find also a database D2 for S2 that “contains the same data” as D1

Run-time support to schema and data translation

- Given
  - a database D1 for a schema S1 in model M1
  - and model M2
- let D1 be accessed as if it were in a schema S2 in model M2
  - so, S2 is again the translation of S1 into M2

A metamodel approach

- The constructs in the various models are rather similar:
  - can be classified into a few categories (Hull & King 1986):
    - Abstract (entity, class, …)
    - Lexical: set of printable values (domain)
    - Aggregation: a construction based on (subsets of) cartesian products (relationship, table)
    - Function (attribute, property)
    - Hierarchies
    - …
- We can fix a set of metaconstructs (each with variants):
  - abstract, lexical, aggregation, function, ...
  - the set can be extended if needed, but this will not be frequent
- **A model is defined in terms of the metaconstructs it uses**

The metamodel approach, example

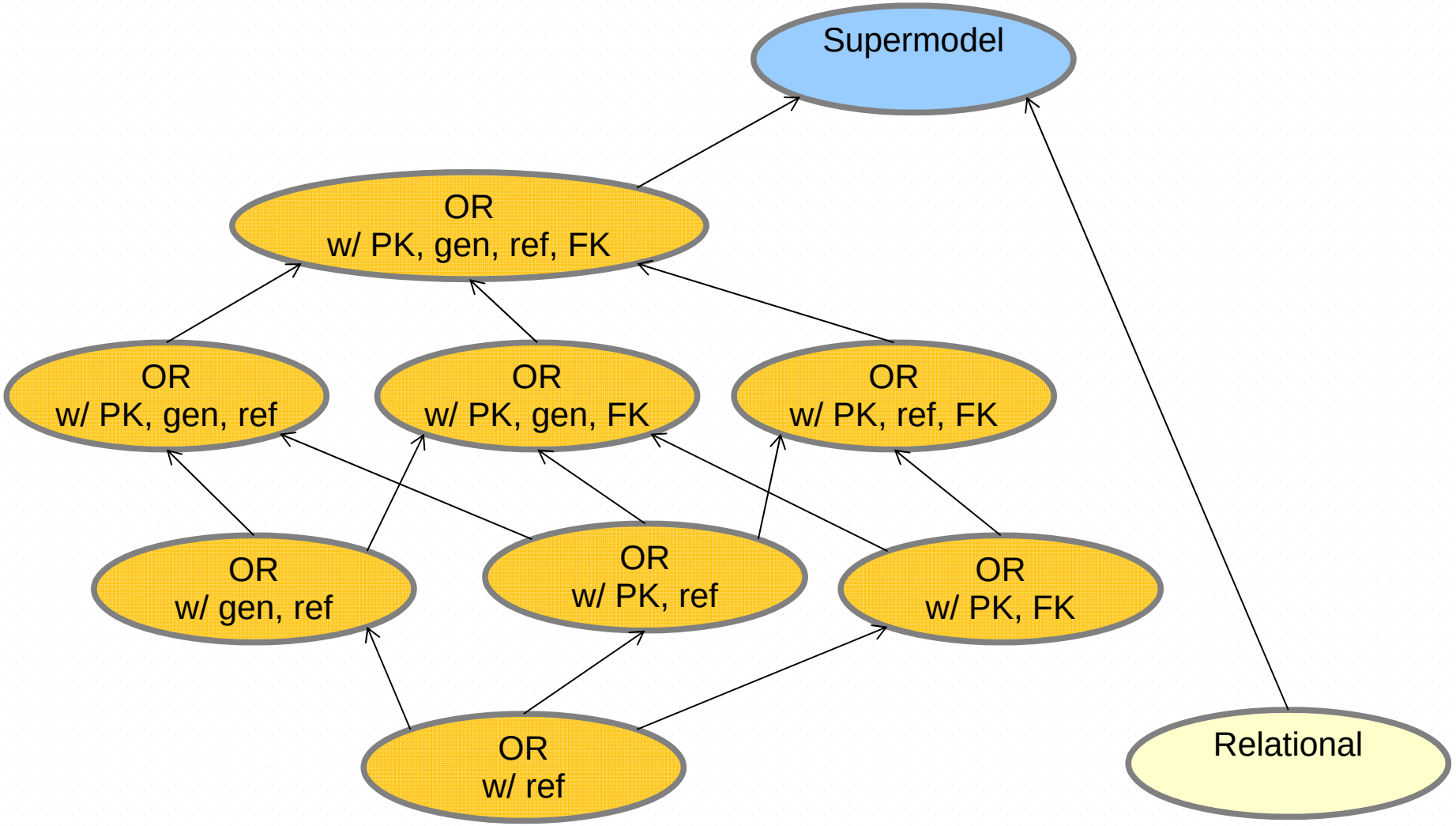
- The ER model:
  - Abstract (called Entity)
  - Function from Abstract to Lexical (Attribute)
  - Aggregation of abstracts (Relationship)
  - …
- The OR model:
  - Abstract (Table with ID)
  - Function from Abstract to Lexical (value-based Attribute)
  - Function from Abstract to Abstract (reference Attribute)
  - Aggregation of lexicals (value-based Table)
  - Component of Aggregation of Lexicals (Column)
  - …

The supermodel

- A model that includes all the meta-constructs (in their most general forms)
  - Each model is subsumed by the supermodel (modulo construct renaming)
  - Each schema for any model is also a schema for the supermodel (modulo construct renaming)
- …
- The supermodel is … the Sanscrit of models

- In the example, a model that generalizes ER, OR and relational

A lattice of models



Translations with the supermodel

- **Each translation from the supermodel SM to a target model M is also a translation from any other model to M:**
  - **given *n* models, we need *n* translations, not n^2**
- We still have too many models:
  - we have few constructs, but each has several independent features which give rise to variants
    - for example, within simple OR model versions,
      - Key may be specifiable or not
      - Generalizations may be allowed or not
      - Foreign keys may be used or not
      - Nesting may be used or not
  - Combining all these, we get hundreds of models!
  - The management of a specific translation for each model would be hopeless

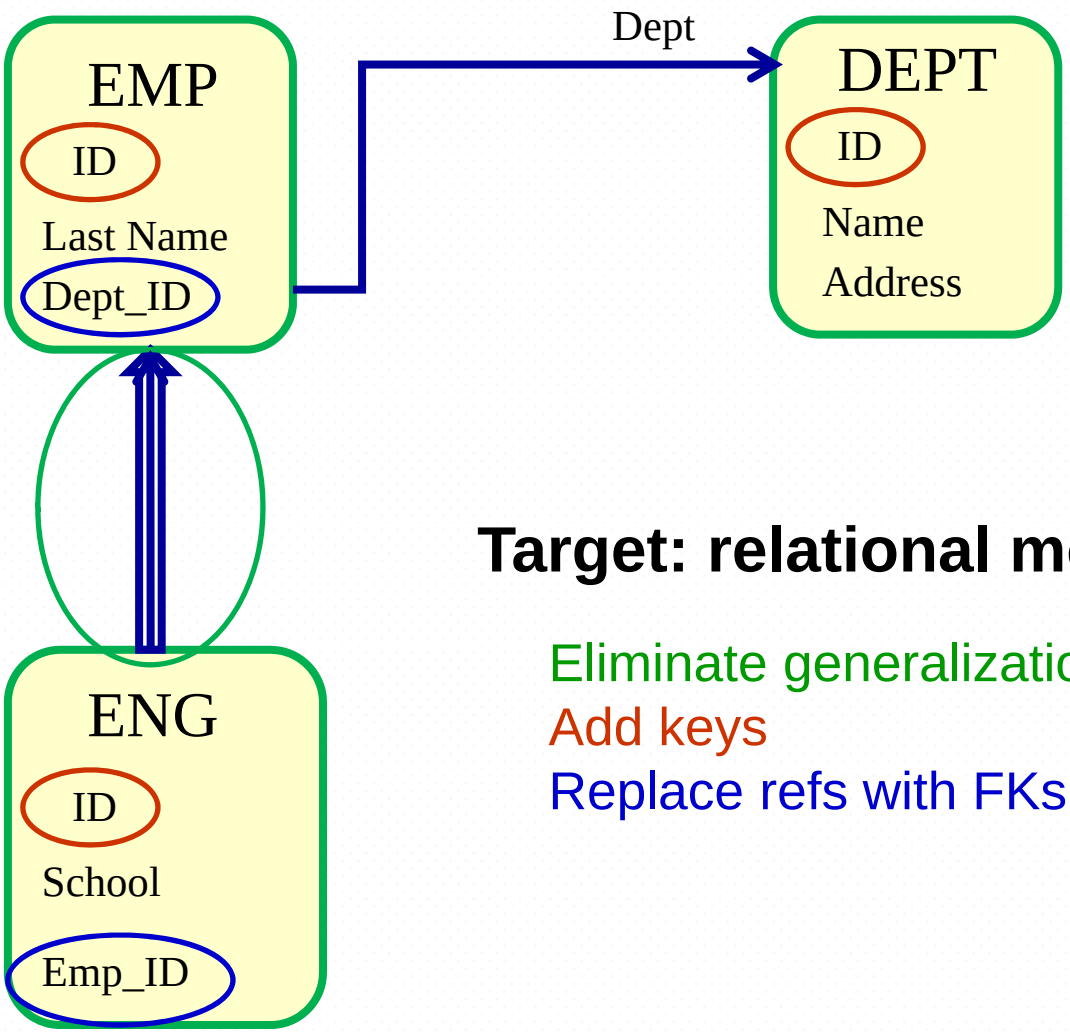
The metamodel approach, translations

- As we saw, the constructs in the various models are similar:
  - can be classified according to the metaconstructs
  - **translations can be defined on metaconstructs,**
    - **there are standard, known ways to deal with translations of constructs (or variants theoreof)**
- Elementary translation steps can be defined in this way
  - Each translation step handles a supermodel construct (or a feature thereof) "to be eliminated" or "transformed"
- Then, elementary translation steps to be combined
- **A translation is the concatenation of elementary translation steps**

An example

- An object relational database, to be translated in a relational one
  - Source: an OR-model
  - Target: the relational model

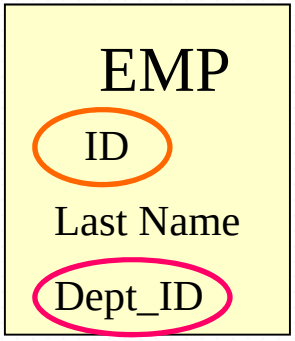
An example, 2

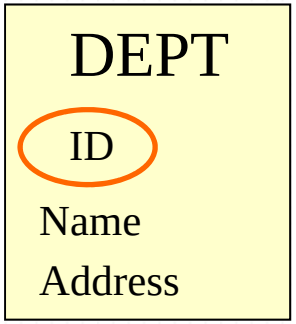


Target: relational model

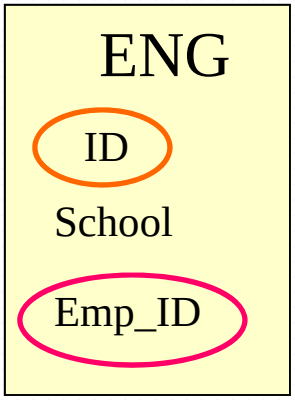
Eliminate generalizations
Add keys
Replace refs with FKs

An example, 3





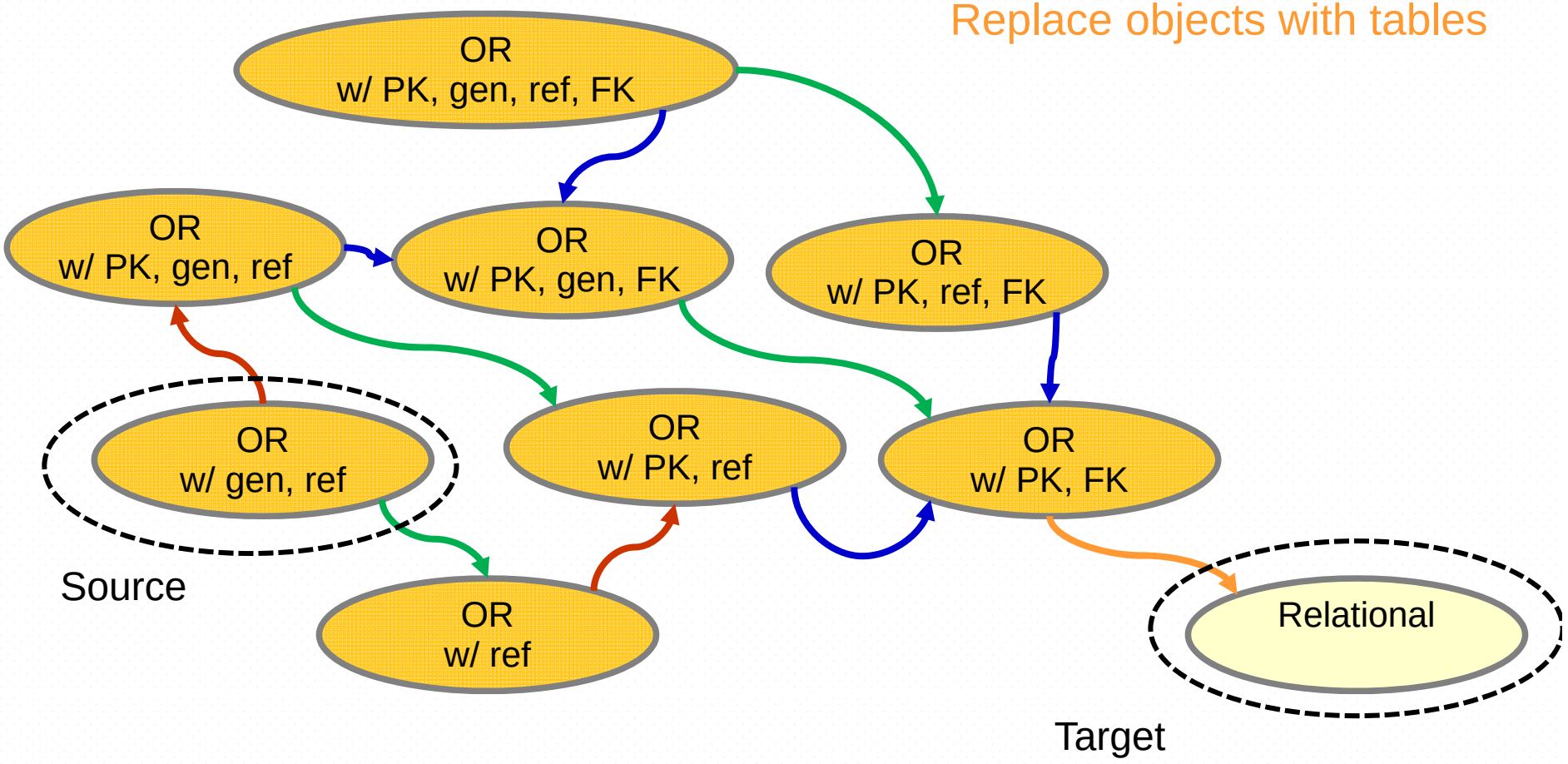
Target: relational model



Eliminate generalizations
Add keys
Replace refs with FKs
Replace objects with tables

The steps for a translation

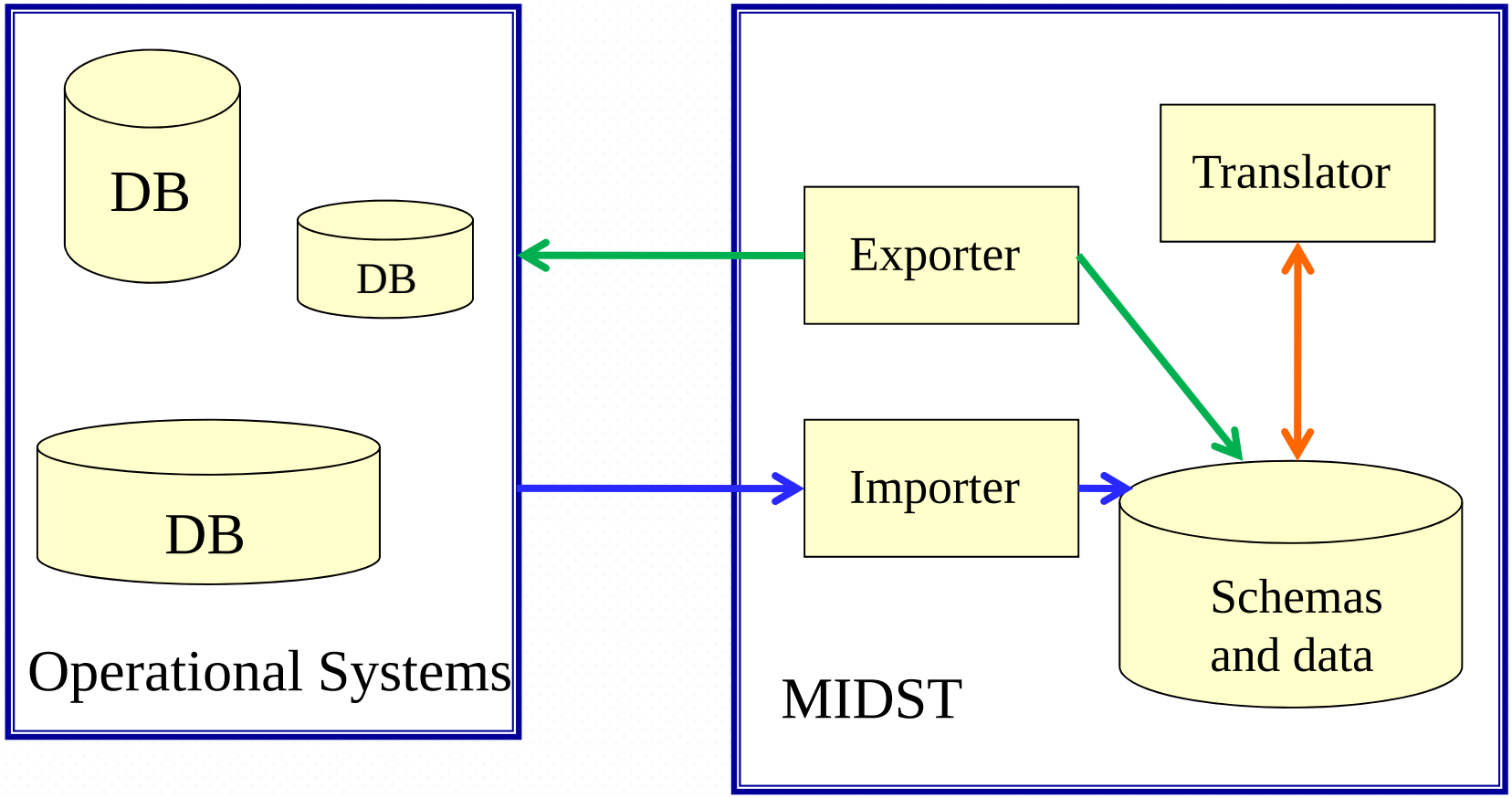
Eliminate generalizations
Add keys
Replace refs with FKs
Replace objects with tables



The MIDST solution

- A metamodel (fixed but extendible)
- A meta dictionary for specifying models within the metamodel
- A dictionary for the description of schemas (and for handling data in the off-line version)
- A library of elementary translations
- An algorithm (and its implementation) for choosing the needed steps given source and target models (based on "signatures" for models and basic translations)

Off-line approach



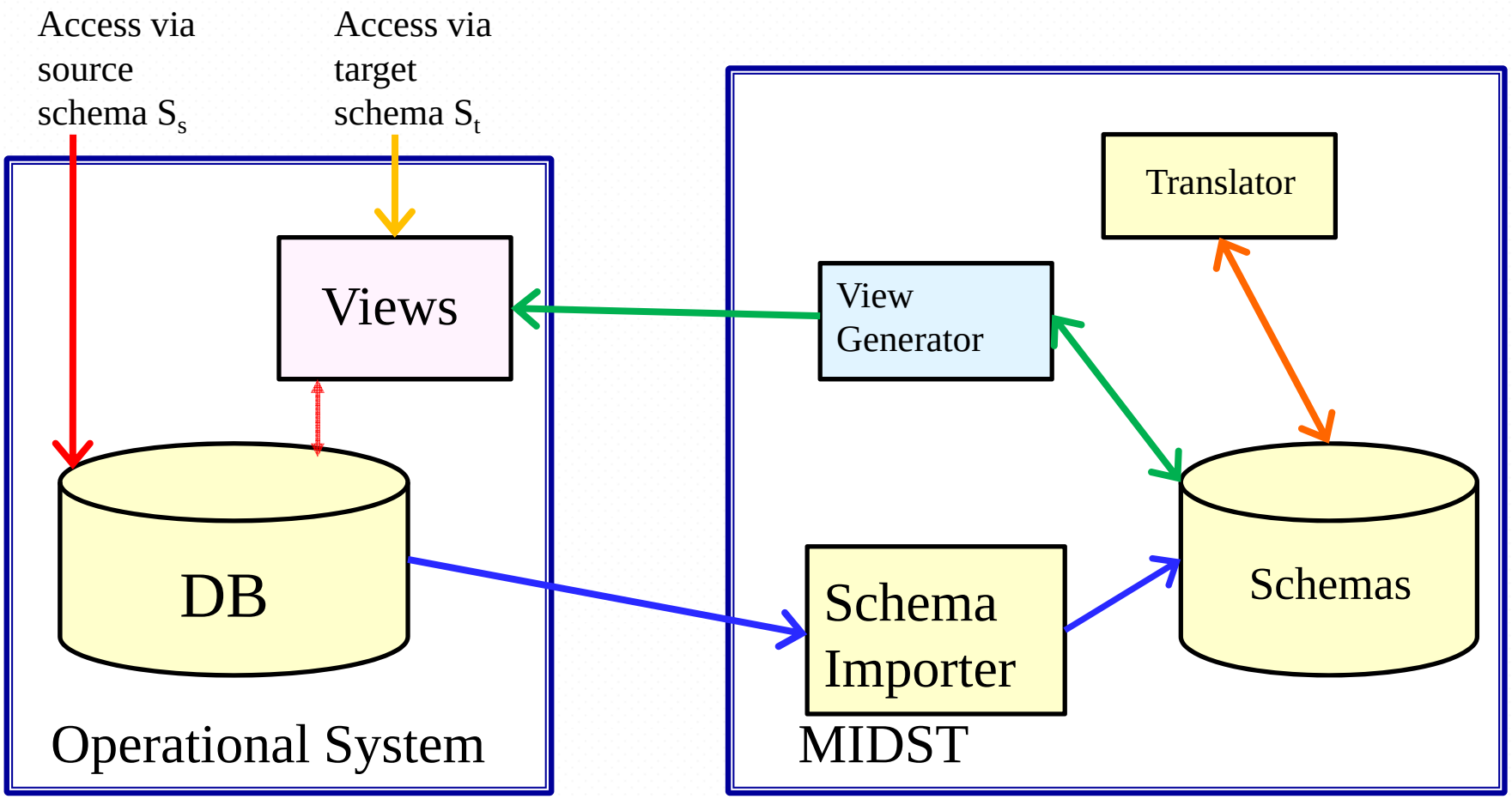
The off-line approach: drawbacks

- Highly inefficient in practice, because it requires databases to be moved back and forth
- It does not allow data to be used directly
- A "run-time" approach is needed

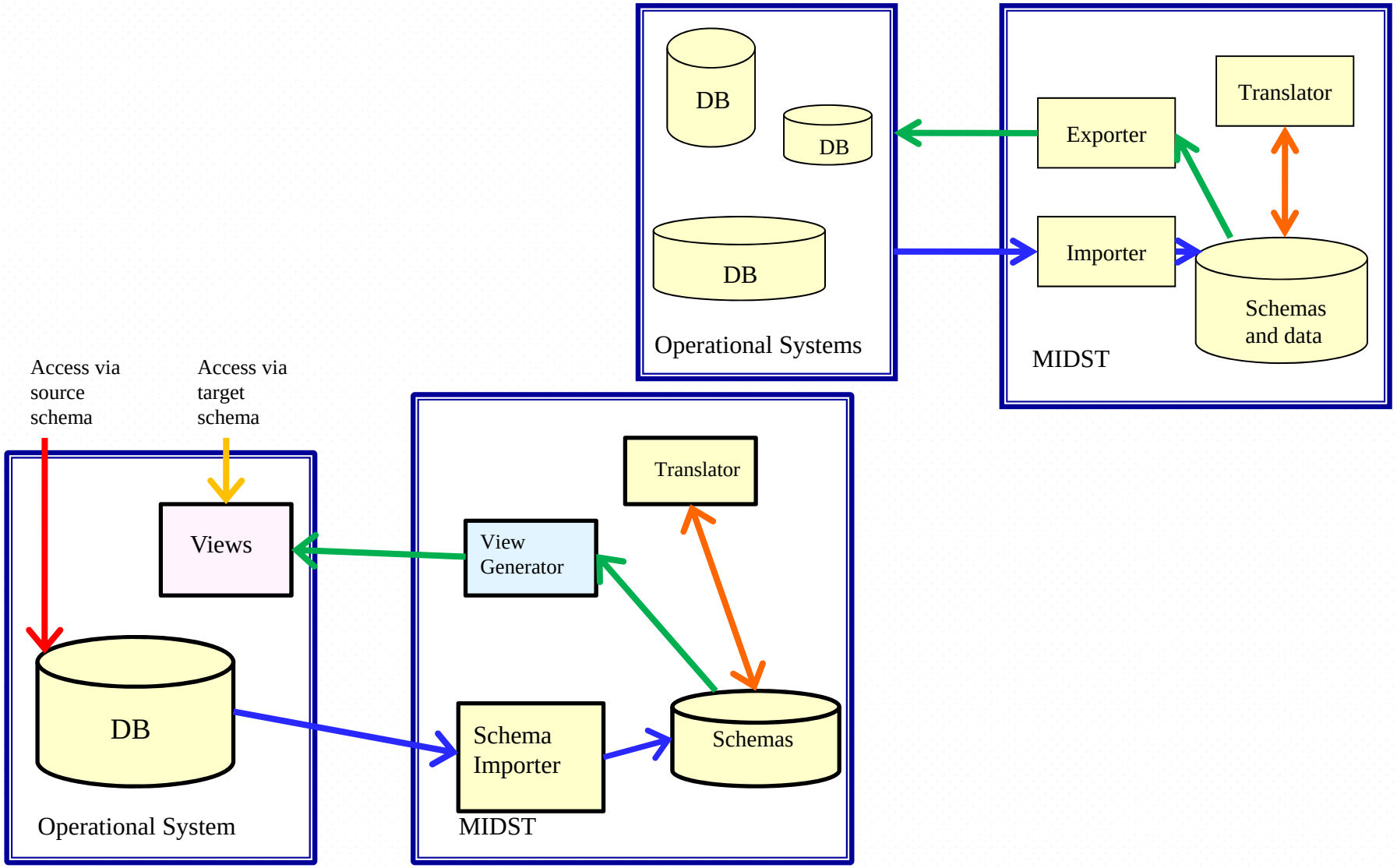
A run-time alternative: generating views

- Main feature:
  - generate, from the datalog traslation programs, executable statements defining views representing the target schema.
- How:
  - by means an analysis of the datalog schema rules under a new classification of constructs

Runtime translation



Run-time vs off-line



Outline

- ✓ Heterogeneity in interoperabilty and migration settings
- ✓ MIDST: heterogeneity and translation with traditional models
- Heterogeneity in NoSQL systems
- A common interface for NoSQL systems
- Future work

"NoSQL" systems

- New family of database systems
  - High scalability (wrt simple operations on many nodes)
  - Replication and distribution (over many nodes)
  - Flexibility in data structure
  - New indexing techniques
  - …
- With some limitations
  - Data model (and API) much simpler than SQL
  - Less strict transaction management

There are many "NoSQL" systems

- "*NoSQL is about choice" (*Jan Lehnardt on April 9, 2010, ages ago in this area, but still valid):
  - no "one-size-fit-all"
  - it could even make sense to use multiple systems within a single application
- Various categories
  - Key-value stores (Redis, Project Voldemort, …)
  - Document stores (SimpleDB, CouchDB, MongoDB, …)
  - Extensible record stores (BigTable, HBase, …)
- Issues:
  - there is no standard (not even an idea …)
  - (comparison of) performances are yet to be understood
  - new systems appear, investments can be wasted (lock-in)

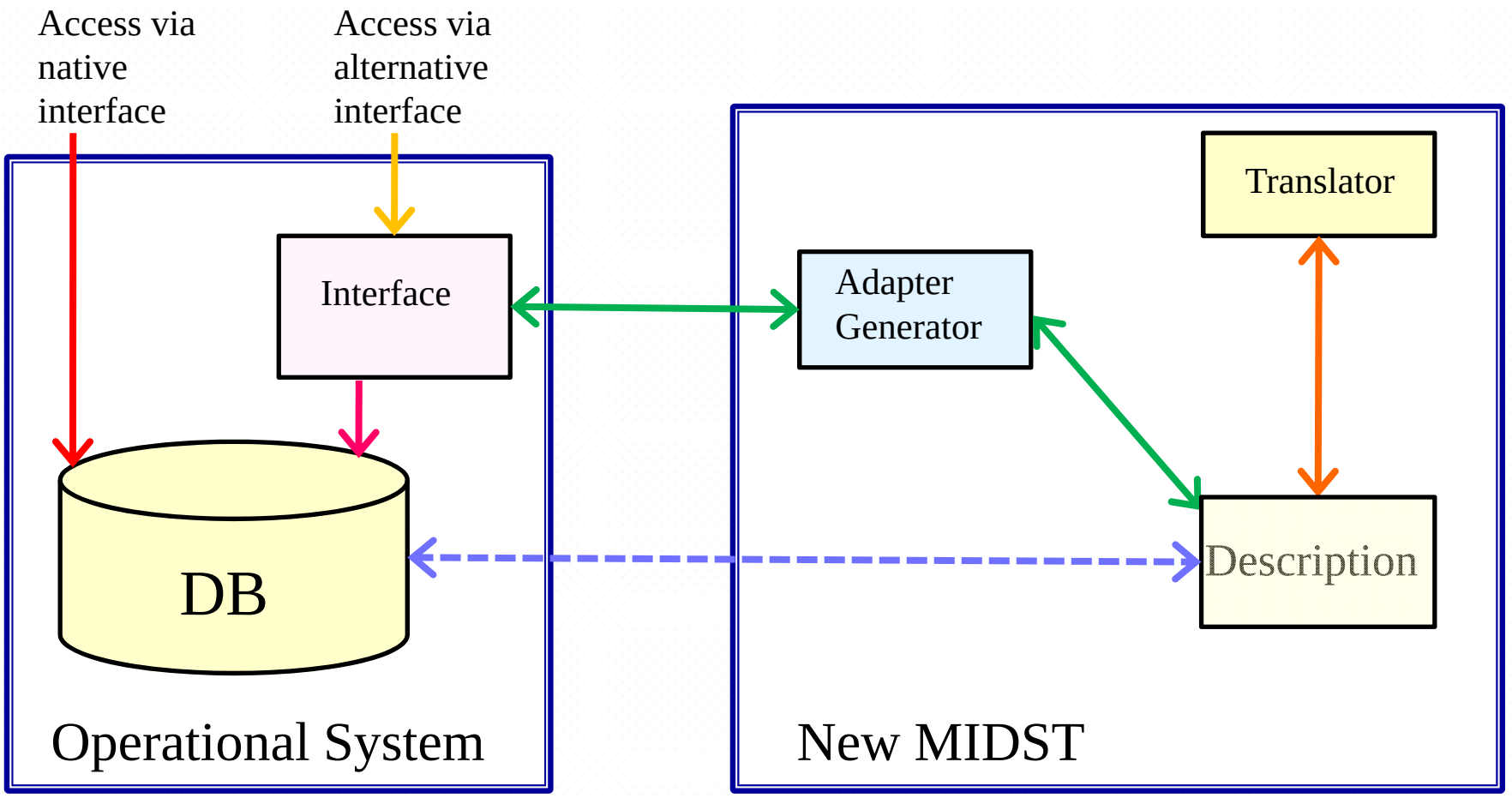
A problem we know

- In the same way as with traditional databases, we have heterogeneity
  - even more than we were used to (even more than with XML)

Which specific version of the problem?

- Schema translation?
  - There is no common notion of schema, so this is not really an issue
- Offline data conversion?
  - Possibly relevant, but there would be need for managing the transition period and also it would not easily support multiple systems
- Runtime support?
  - Important, in order to be able to use the various systems interchangeably and multiple systems at the same time

A long term goal: Runtime translation



Issues

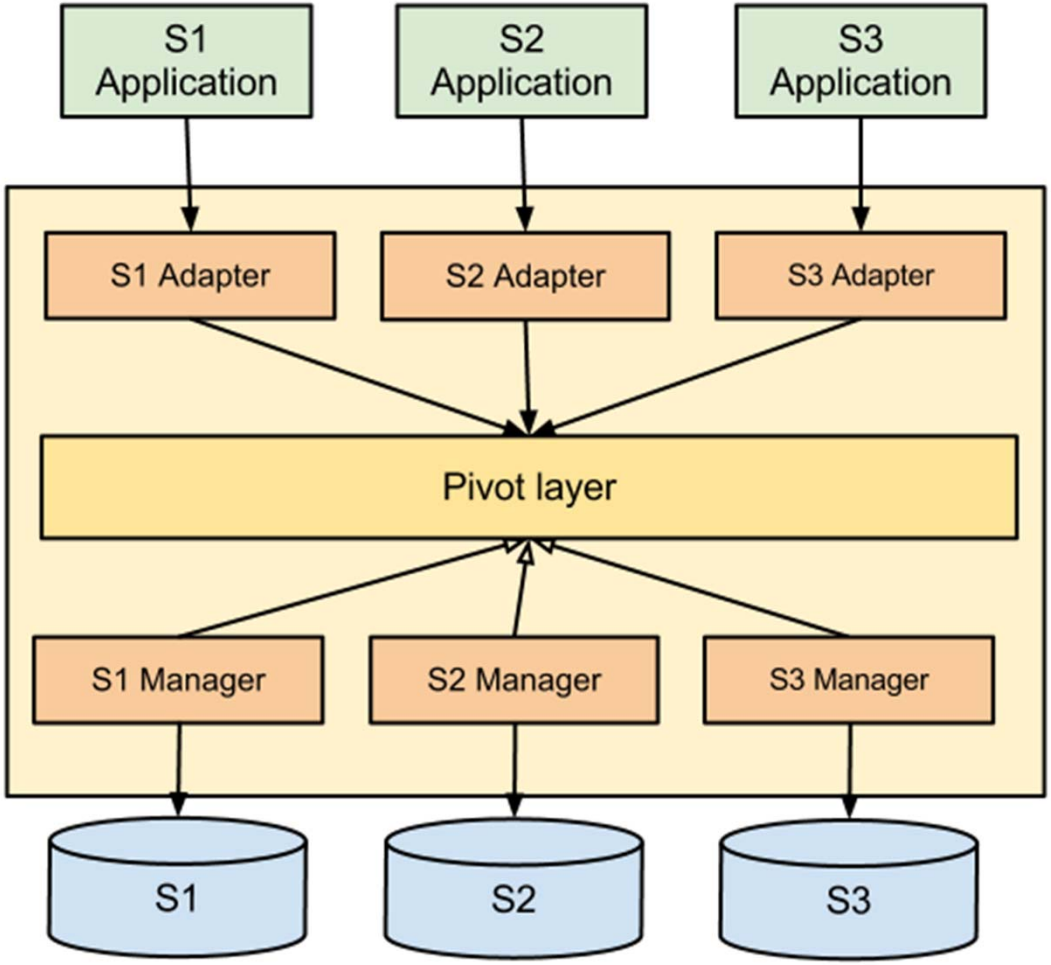
- Settings are not that much similar to those for traditional databases
  - Interfaces
    - are usually much simpler
    - have different "expressive power"
  - The structure of data is represented only to a certain extent (there is no notion of schema, and structure is usually very flexible)
  - Similarly, there is no notion of query language, nor a general pattern for queries

A supermodel based approach?

- In traditional settings, our idea was to have the supermodel as the most general model, at the top of a lattice
- Here, simplicity is a goal, even if objects could have some structure
- Also, while in databases data are "exposed" in full (and so there are powerful query languages that can exploit the structure), here operations are more focussed

- Therefore, while in our previous approach we used as a "pivot" a very rich model, the supermodel, here a much simpler one would be needed
- …
- … Sanscrit would not be much suitable here

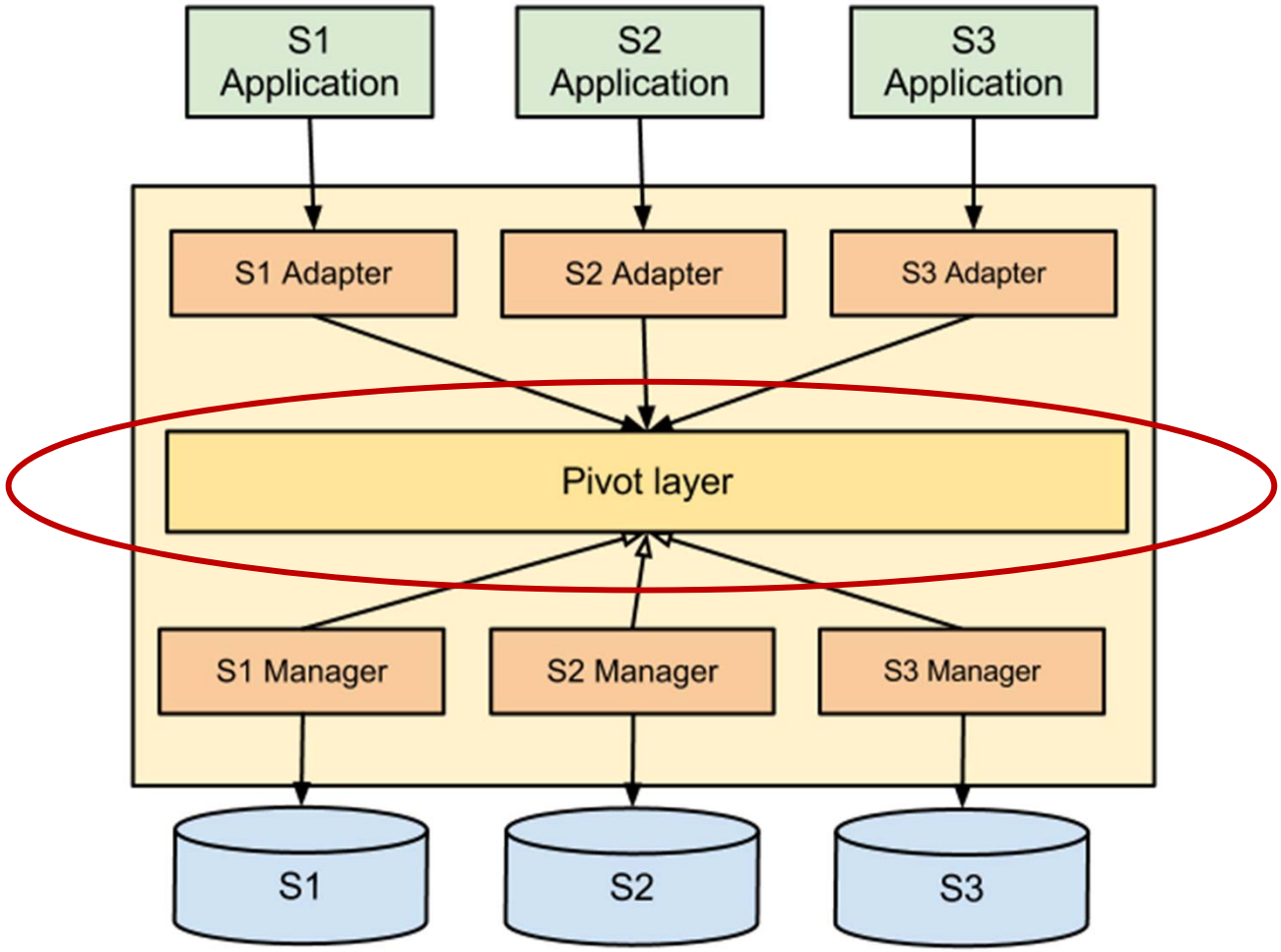
A possible architecture



Outline

- ✓ Heterogeneity in interoperabilty and migration settings
- ✓ MIDST: heterogeneity and translation with traditional models
- ✓ Heterogeneity in NoSQL systems
- A common interface for NoSQL systems
- Future work

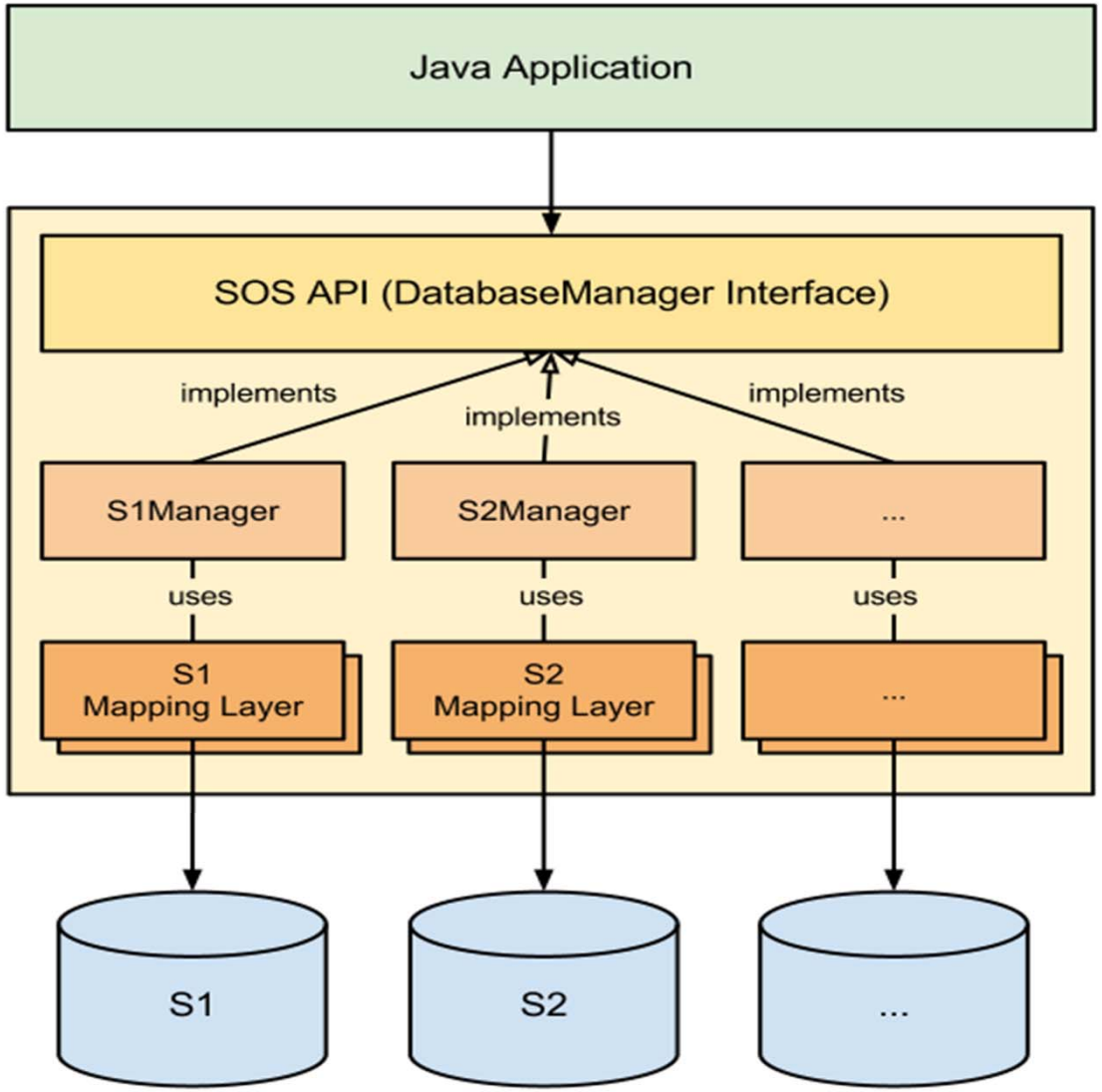
A possible architecture



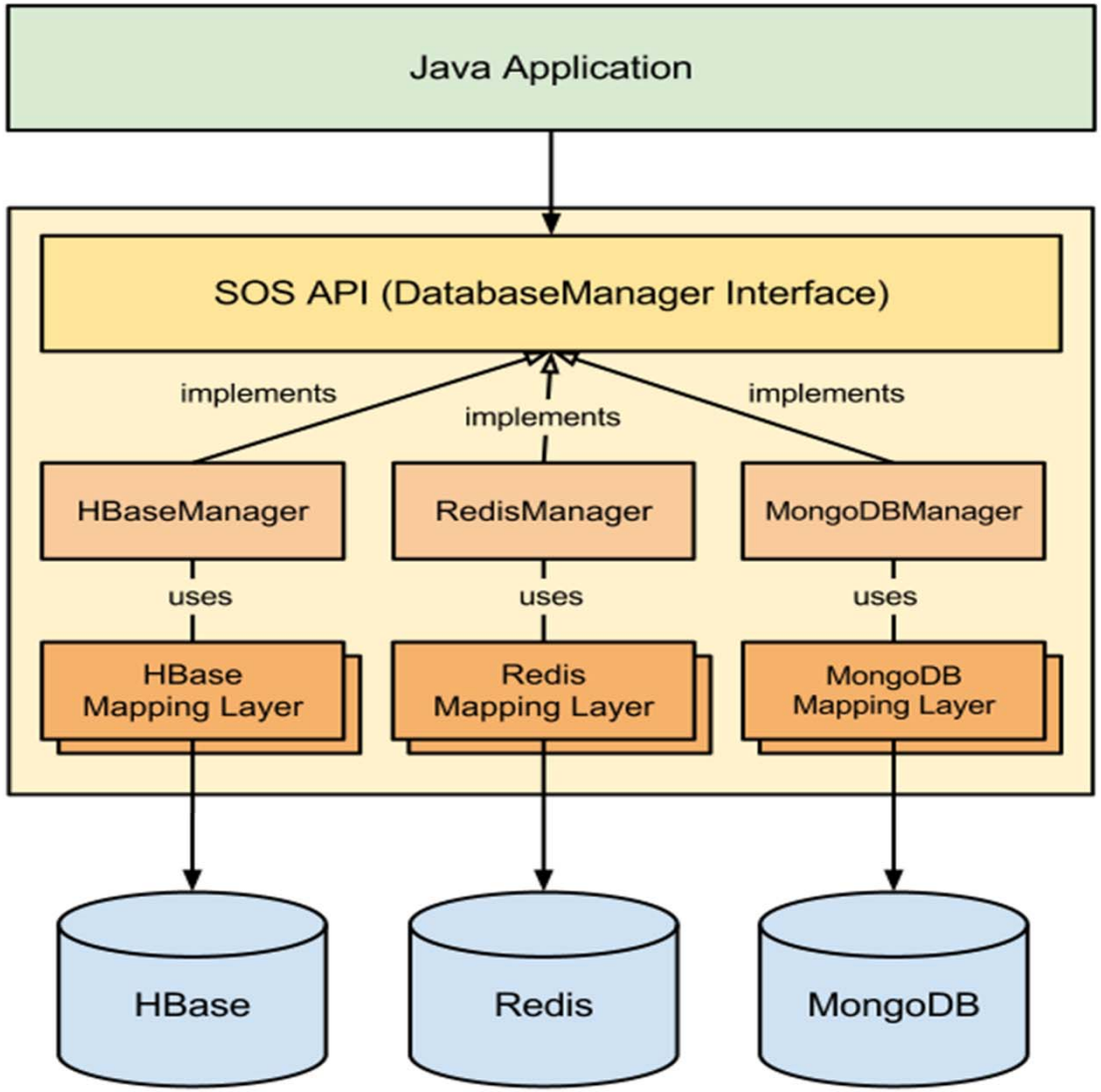
A first step

- A common interface, with a simple set of methods involving single objects or (for retrieval) sets thereof
  - put
  - get
  - delete
- Motivation
  - the general, common goal of NoSQL systems is to support simple operations
- First implementation in Java

SOS: Save Our Systems



SOS, concretely



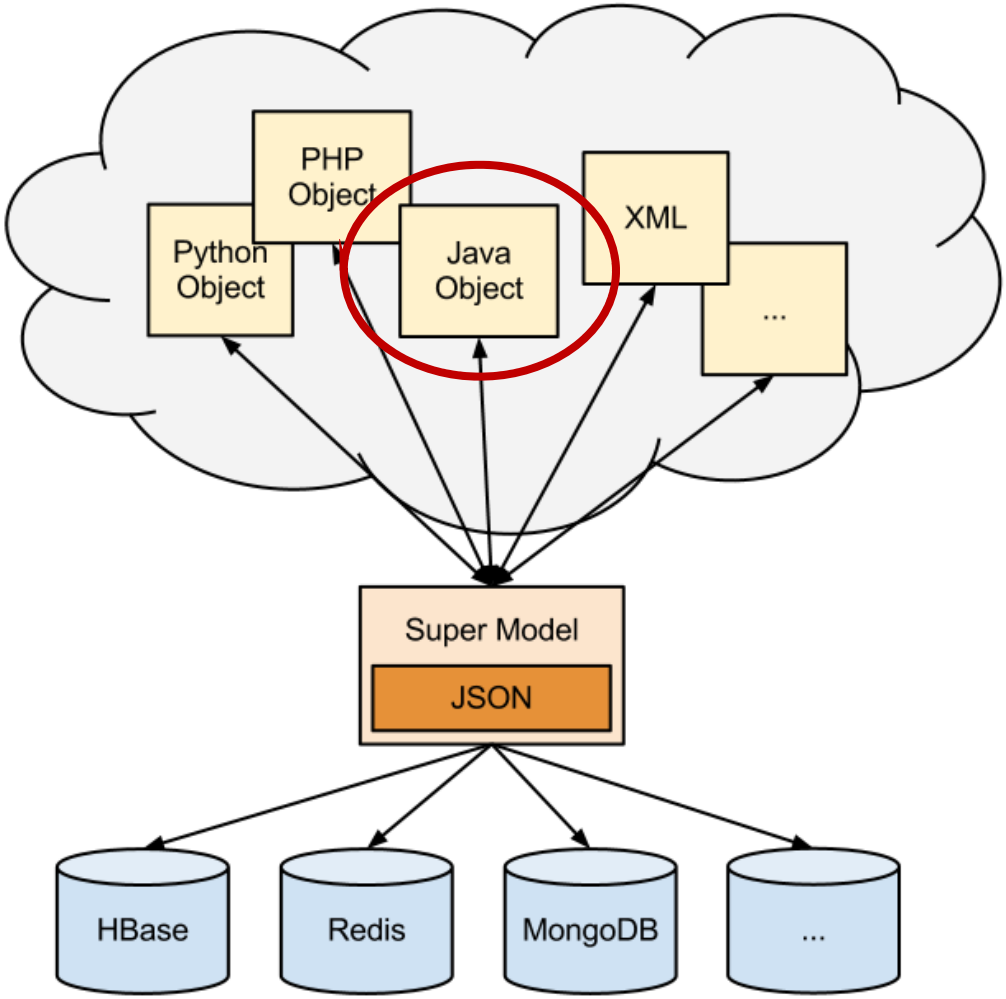
Issues

- Do objects have a structure? Should we handle it?
- How much sophisticated is the retrieval (get) operation?

Object structure

- In general, to support a very basic interface, we could just treat objects as blobs, serializing them
- However, objects often have a complex structure, which can be modeled in tree form, with sets and structures, possibly nested, as well as simple attributes
- Our interface gets the native objects and the implementation serializes them into JSON

Implementation of the structure



The get operation

- Various forms in mind
  1. `Object get (String collection, String ID)`
  2. `Object get (String collection, Path p)`
  3. `Set<Object> get (Query q)`
- Currently the first two implemented
  1. Straightforward
  2. Currently retrieval of simple fields, in the future reconstruction of objects
  3. Many interesting challenges, related to query processing and performances

Outline

- ✓ Heterogeneity in interoperabilty and migration settings
- ✓ MIDST: heterogeneity and translation with traditional models
- ✓ Heterogeneity in NoSQL systems
- ✓ A common interface for NoSQL systems
- • Future work

The next steps

- With respect to SOS:
  - more implementation features, with flexibility
  - custom mappings
  - performance evaluation
- An interoperability approach:
  - Given the goals of NoSQL systems, a general "metamodel based" approach is not obvious
  - The definition of a supermodel should balance expressivity with simplicity
  - The simple interface we have developed could be a reasonable pivot, limiting the expressive power

Main references

- MIDST, offline approach
  - P. Atzeni, P. Cappellari, R. Torlone, P. A. Bernstein, G. Gianforme: Model-independent schema translation. VLDB J. 17(6): 1347-1370 (2008)
  - P. Atzeni, P. Cappellari, P. A. Bernstein: Model-Independent Schema and Data Translation. EDBT 2006: 368-385
- MIDST, runtime approach
  - P. Atzeni, L. Bellomarini, F. Bugiotti, G. Gianforme: A runtime approach to model-independent schema and data translation. Information Systems, 37(3): 269-287 (2012).
  - P. Atzeni, L. Bellomarini, F. Bugiotti, G. Gianforme: A runtime approach to model-independent schema and data translation. EDBT 2009: 275-286
- SOS
  - P. Atzeni, F., Bugiotti, L. Rossi. Uniform access to non-relational database systems: the SOS platform. CAiSE 2012.
  - P. Atzeni, F., Bugiotti, L. Rossi. SOS (Save Our Systems): A uniform programming interface for non-relational systems. EDBT 2012 (demo section).

Thank you!