Schema and Data Translation

Paolo Atzeni
Università Roma Tre

Based on

Paper in EDBT 1996 (with R. Torlone)
Paper in EDBT 2006 (with P. Cappellari and P. Bernstein)
Demo in Sigmod 2007 (with P. Cappellari and G. Gianforme)

InfInt -- Bertinoro, October 4, 2007

Terminology: a warning

Model Mgmt people	Traditional DB people
Meta-metamodel	Metamodel
Metamodel	Model
Model	Schema

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” to D1

A long standing issue

- Translations from a model to another have been studied since the 1970's
- Whenever a new model is defined, techniques and tools to generate translations are studied
- However, proposals and solutions are usually model specific:
  - Given an ER schema, find the suitable relational schema that "implements" it
    - the original paper (Chen 1976) contains the basics
    - further discussions by many (e.g. Markowitz and Shoshani 1989)
    - illustrated in every textbook
  - Similarly with
    - any other conceptual model and any other logical one
    - XML and relational (or object)

We have been doing this for a while

- Initial work more than ten years ago (Atzeni & Torlone, 1996)
- Major novelty recently (Atzeni, Cappellari & Bernstein, 2006; Atzeni, Cappellari & Gianforme, 2007)
  - translation of both schemas **and data**
  - data-level translations generated automatically, from schema-level ones

A simple example

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

Employees		
EmpNo	Name	Dept
134	Smith	●
201	Jones	●
255	Black	●
302	Brown	●

Departments	
Name	Address
A	5, Pine St
B	10, Walnut St

System managed ids used as references

Example, 2

Employees

	EmpNo	Name	Dept
E#1	134	Smith	D#1
E#2	201	Jones	D#2
E#3	255	Black	D#1
E#4	302	Brown	null

Departments

	Name	Address
D#1	A	5, Pine St
D#2	B	10, Walnut St

- Does the OR model allow for keys?
- Assume EmpNo and Name are keys

Employees

EmpNo	Name	Dept
134	Smith	A
201	Jones	B
255	Black	A
302	Brown	null

Departments

Name	Address
A	5, Pine St
B	10, Walnut St

Example, 3

	Employees		
	EmpNo	Name	Dept
E#1	134	Smith	D#1
E#2	201	Jones	D#2
E#3	255	Black	D#1
E#4	302	Brown	null

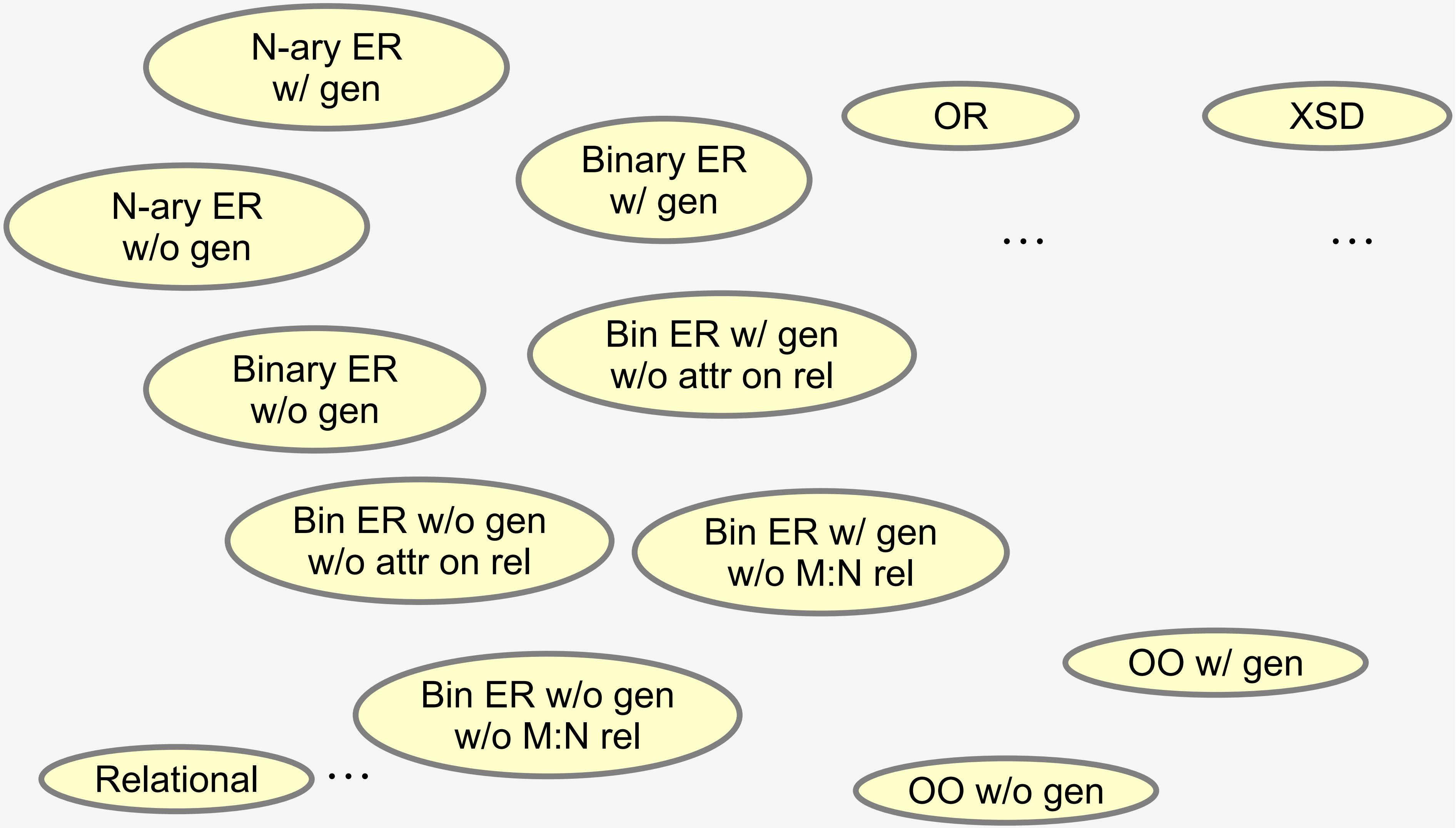
	Departments	
	Name	Address
D#1	A	5, Pine St
D#2	B	10, Walnut St

- Does the OR model allow for keys?
- Assume no keys are specified

Employees			
EmpID	EmpNo	Name	Dept
1	134	Smith	1
2	201	Jones	2
3	255	Black	1
4	302	Brown	null

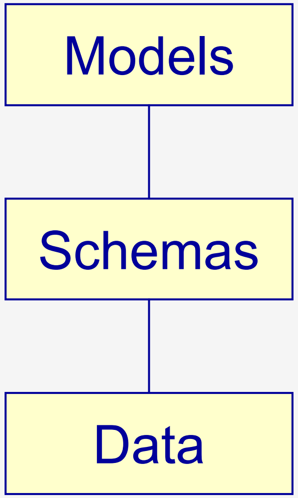
Departments		
DeptID	Name	Address
1	A	5, Pine St
2	B	10, Walnut St

Many different models (and variations …)



Heterogeneity

- We need to handle artifacts and data in various models
  - Data are defined wrt to schemas
  - Schemas are defined wrt to models
  - How models can be defined?



A metamodel approach

- The constructs in the various models are rather similar:
  - can be classified into a few categories (Hull & King 1986):
    - Lexical: set of printable values (domain)
    - Abstract (entity, class, …)
    - 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):
  - lexical, abstract, 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)
  - …

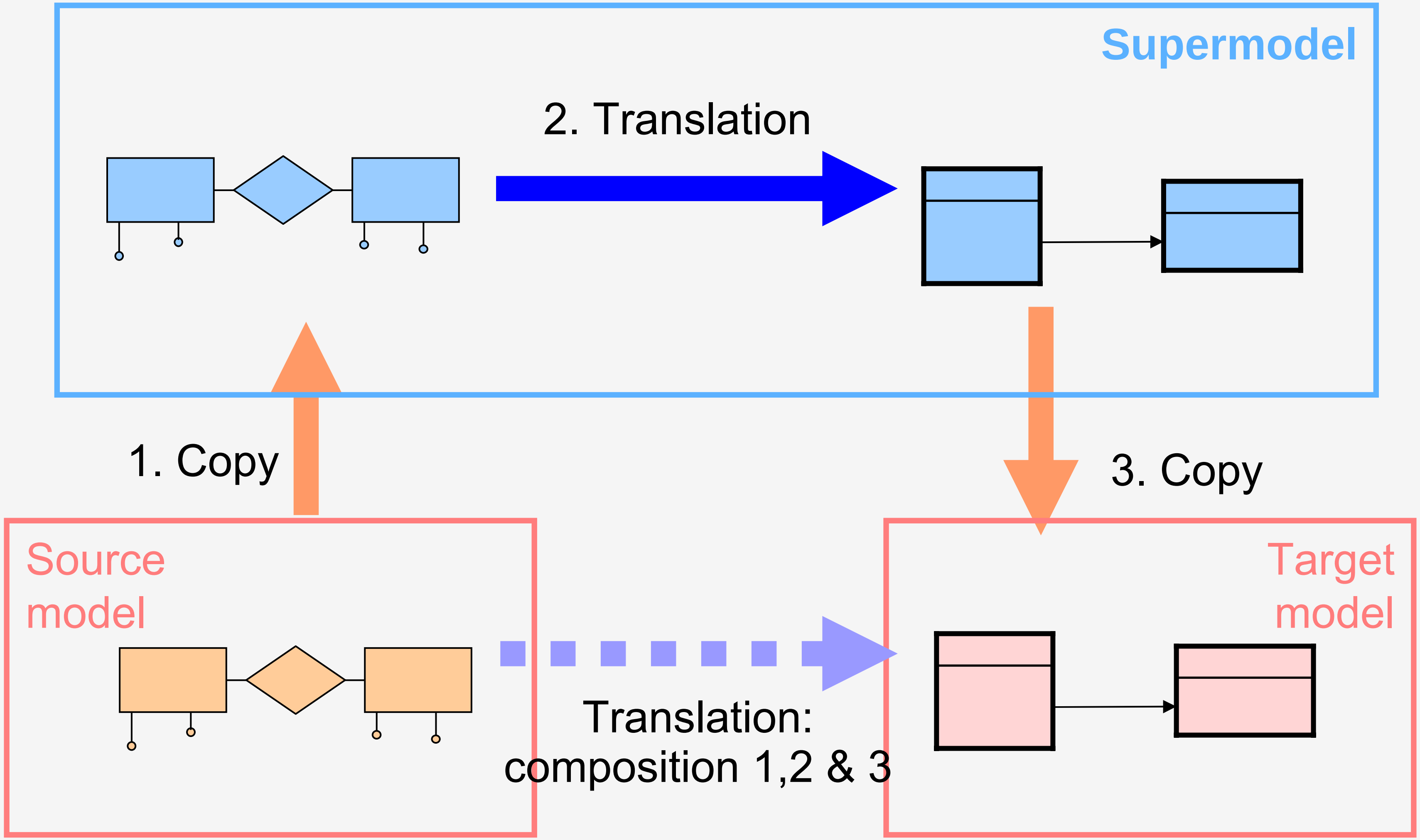
The supermodel

- 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 metamodel approach, translations

- The constructs in the various models are rather similar:
  - can be classified into a few categories (“metaconstructs")
  - translations can be defined on metaconstructs,
    - and there are “standard”, accepted ways to deal with translations of metaconstructs
    - they can be performed within 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

Generic translation environment



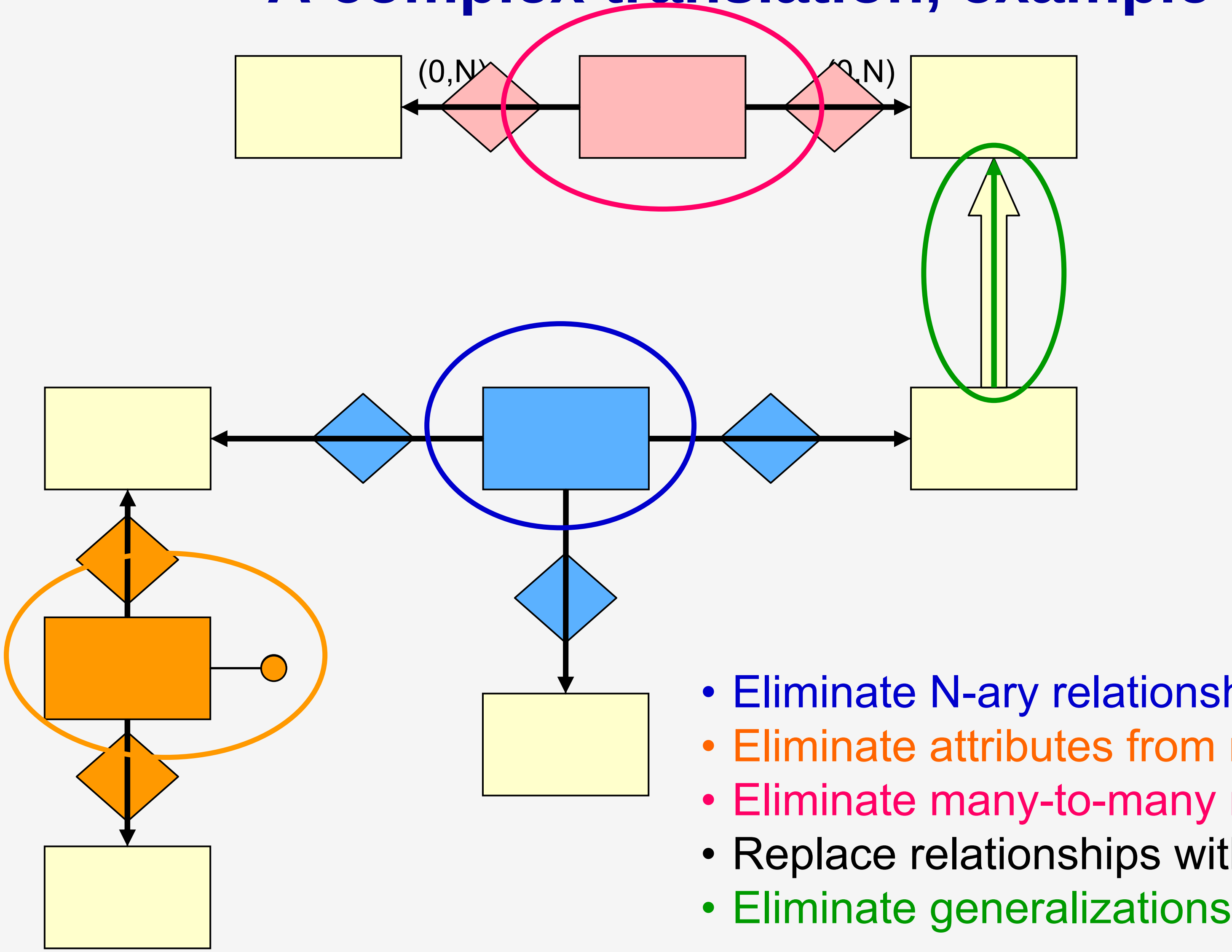
Translations within the supermodel

- We still have too many models:
  - Just within simple ER model versions, we have 4 or 5 constructs, and each has several independent features which give rise to variants
    - for example, relationships can be
      - binary or N-ary
      - with all possible cardinalities or without many-to-many
      - with or without the possibility of specifying optionality
      - with or without attributes
      - …
  - Combining all these, we get hundreds of models!
  - The management of a specific translation for each model would be hopeless

Translations, the approach

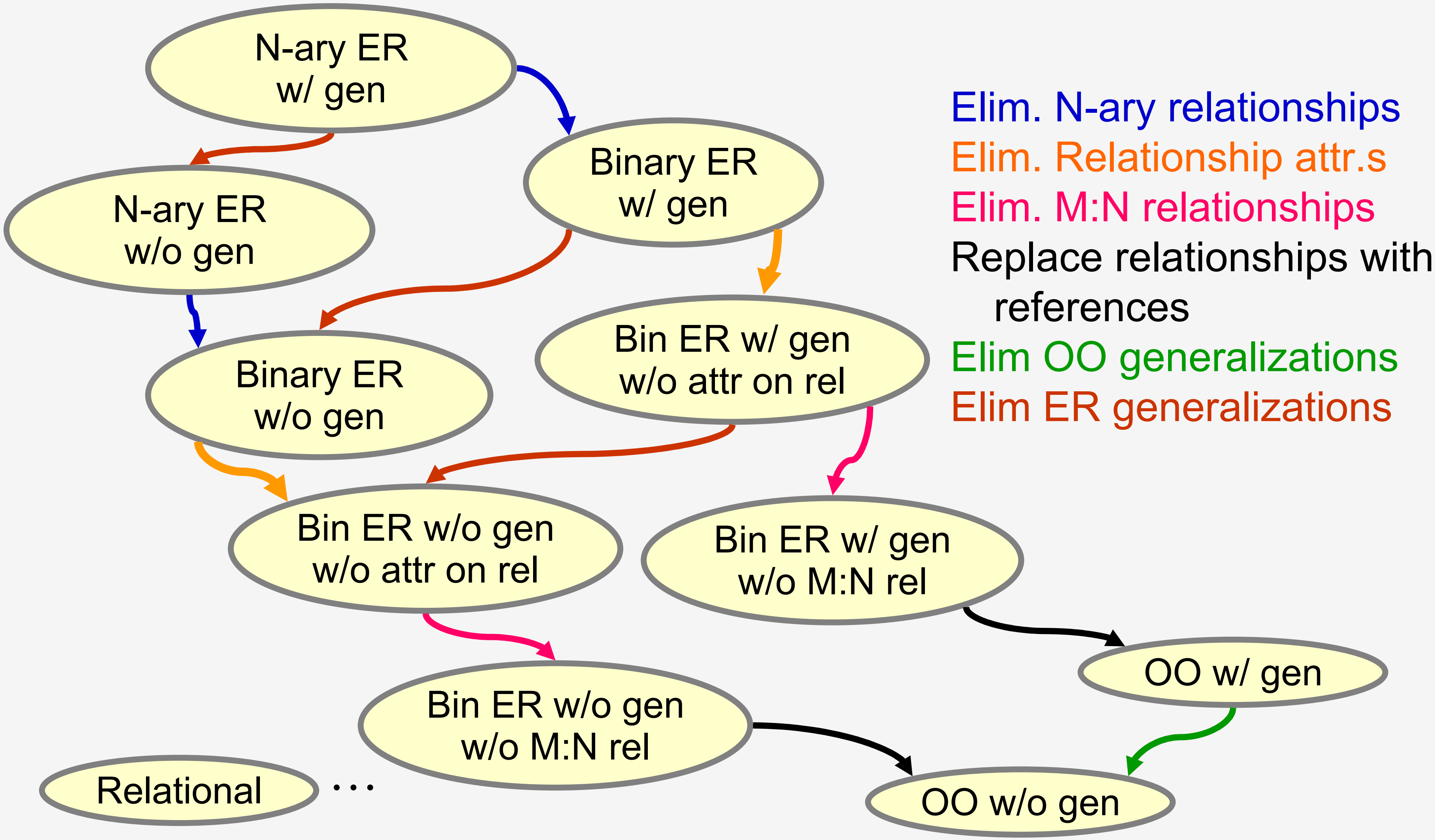
- Elementary translation steps to be combined
- Each translation step handles a supermodel construct (or a feature thereof) "to be eliminated" or "transformed"
- A translation is the concatenation of elementary translation steps

A complex translation, example



- Eliminate N-ary relationships
- Eliminate attributes from relationships
- Eliminate many-to-many relationships
- Replace relationships with references
- Eliminate generalizations

Complex translations



Translations

- Basic translations are written in a variant of Datalog, with OID invention
  - We specify them at the schema level
  - The tool "translates them down" to the data level
  - Some completion or tuning may be needed

A Multi-Level Dictionary

- Handles models, schemas and data
- Has both a model specific and a model independent component
- Relational implementation, so Datalog rules can be easily specified

Multi-Level Dictionary

description

	model specific	model generic
model	Model descriptions (mM)	Supermodel description (mSM)
schema	Model specific schemas (M)	Supermodel schemas (SM)
data	Model specific instances (i-M)	Supermodel instances (i-SM)

model independence

mM	mSM
M	SM
i-M	i-SM

The supermodel description

MSM-Construct

OID	Name	IsLex
1	AggregationOfLexicals	F
2	ComponentOfAggrOfLex	T
3	Abstract	F
4	AttributeOfAbstract	T
5	BinaryAggregationOfAbstracts	F

MSM-Reference

OID	Name	Construct	Target
30	Aggregation	2	1
31	Abstract	4	3
32	Abstract1	5	3
33	Abstract2	5	3

MSM-Property

OID	Name	Constr.	Type
11	Name	1	String
12	Name	2	String
13	IsKey	2	Boolean
14	IsNullable	2	Boolean
15	Type	2	String
16	Name	3	String
17	Name	4	String
18	IsIdentifier	4	Boolean
19	IsNullable	4	Boolean
20	Type	4	String
21	IsFunct1	5	Boolean
22	IsOptional1	5	Boolean
23	Role1	5	String
24	IsFunct2	5	Boolean
25	IsOptional2	5	Boolean
26	Role2	5	String

mM	mSM
M	SM
i-M	i-SM

Model descriptions

MM-Model

<u>OID</u>	Name
1	Relational
2	Entity-Relationship
3	Object

MM-Construct

<u>OID</u>	Model	MSM-Constr	Name
1	2	3	ER_Entity
2	2	4	ER_Attribute
3	2	5	ER_Relationship
4	1	1	Rel_Table
5	1	2	Rel_Column
6	3	3	OO_Class

MM-Property

…	…	…	…

MM-Reference

…	…	…	…

MSM-Construct

<u>OID</u>	Name	IsLex
1	AggregationOfLexicals	F
2	ComponentOfAggrOfLex	T
3	Abstract	F
4	AttributeOfAbstract	T
5	BinaryAggregationOfAbstracts	F

MSM-Property

<u>OID</u>	Name	Construct	Type
…	…	…	…

MSM-Reference

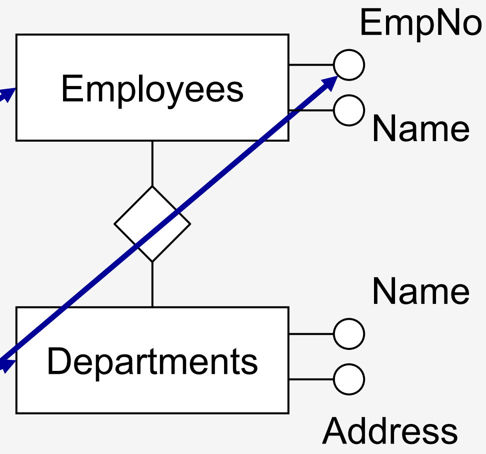
<u>OID</u>	Name	Construct	…
…	…	…	…

mM	mSM
M	SM
i-M	i-SM

Schemas in a model

SM-Construct

OID	Name	IsLex
…	…	…
3	ER-Entity	F
4	ER-AttributeOfEntity	T
…	…	…



ER-Entity

OID	Schema	Name
301	1	Employees
302	1	Departments
201	3	Clerks
202	3	Offices

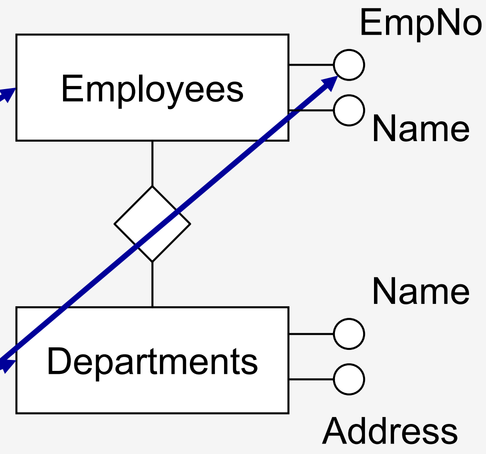
ER-AttributeOfEntity

OID	Schema	Name	isIdent	isNullable	Type	AbstrOID
401	1	EmpNo	T	F	Int	301
402	1	Name	F	F	Text	301
404	1	Name	T	F	Char	302
405	1	Address	F	F	Text	302
501	3	Code	T	F	Int	201
…	…	…	…	…	…	…

ER schemas

mM	mSM
M	SM
i-M	i-SM

Schemas in the supermodel



MSM-Construct

OID	Name	IsLex
…	…	…
3	Abstract	F
4	AttributeOfAbstract	T
…	…	…

SM-Abstract

OID	Schema	Name
301	1	Employees
302	1	Departments
201	3	Clerks
202	3	Offices

SM-AttributeOfAbstract

OID	Schema	Name	isIdent	isNullable	Type	AbstrOID
401	1	EmpNo	T	F	Int	301
402	1	Name	F	F	Text	301
404	1	Name	T	F	Char	302
405	1	Address	F	F	Text	302
501	3	Code	T	F	Int	201
…	…	…	…	…	…	…

Supermodel schemas

mM	mSM
M	SM
i-M	i-SM

Instances in the supermodel

SM-Abstract		
OID	Schema	Name
301	1	Employees
302	1	Departments
201	3..	Clerks
202	2	Offices

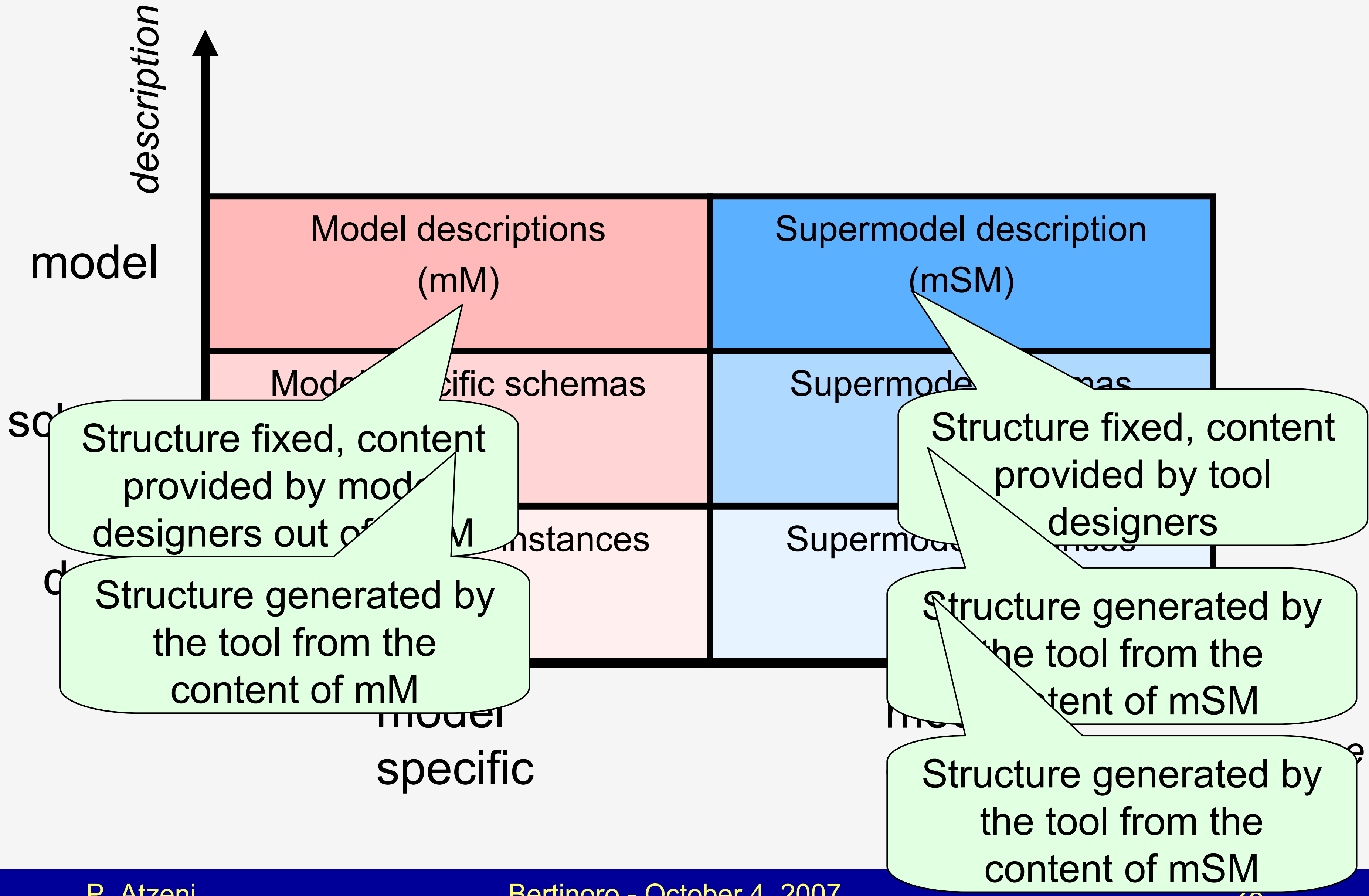
OID	AbsOID	InstOID
3010	301	1
3011	302	1
3012	301	1

SM-AttributeOfAbstract						
OID	Schema	Name	isIdent	isNullable	Type	AbstrOID
401	1	EmpNo	T	F	Int	301
402	1	Name	F	F	Text	301
404	1	Name	T	F	Char	302
405	1	Address	F	T	Text	302
501	3	Code	T	F	Int	201
…	…	…	…	…	…	…

OID	AttrOfAbsOID	InstOID	Value	i-AbsOID
4010	401	1	75432	3010
4020	402	1	John Doe	3010
4021	402	1	Bob White	3011
…	…	…	…	…

Supermodel instances

Multi-Level Repository, generation and use



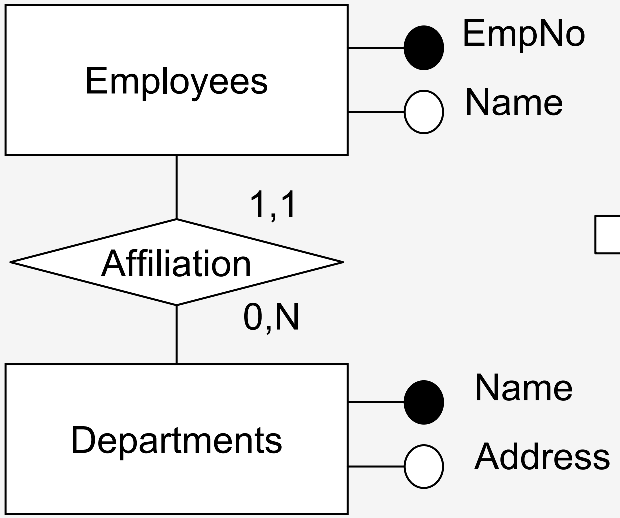
Translations

- Basic translations are written in a variant of Datalog, with OID invention
  - We specify them at the schema level
  - The tool "translates them down" to the data level
  - Some completion or tuning may be needed

A basic translation

- From (a simple) binary ER model to the relational model
  - a table for each entity
  - a column (in the table for E) for each attribute of an entity E
  - for each M:N relationship
    - a table for the relationship
    - columns …
  - for each 1:N and 1:1 relationship:
    - a column for each attribute of the identifier …

A basic translation application



Employees		
EmpNo	Name	Affiliation

Departments	
Name	Address

A basic translation (in supermodel terms)

- From (a simple) binary ER model to the relational model
  - a table for each entity
  - a column (in the table for E) for each attribute of an entity E
  - for each M:N relationship
    - a table for the relationship
    - columns …
  - for each 1:N and 1:1 relationship:
    - a column for each attribute of the identifier …

an aggregation of lexicals for each abstract

a component of the aggregation for each attribute of abstract

aggregation of abstracts …

…

"An aggregation of lexicals for each abstract"

```
SM_AggregationOfLexicals(
    OID: #aggregationOID_1(OID),
    Name: name)
←
SM_Abstract (
    OID: OID,
    Name: name ) ;
```

Datalog with OID invention

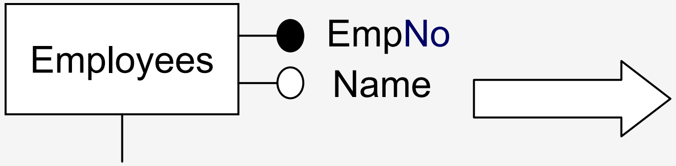
- Datalog (informally):
  - a logic programming language with no function symbols and predicates that correspond to relations in a database
  - we use a non-positional notation
- Datalog with OID invention:
  - an extension of Datalog that uses Skolem functions to generate new identifiers when needed
- Skolem functions:
  - injective functions that generate "new" values (value that do not appear anywhere else); so different Skolem functions have disjoint ranges

"An aggregation of lexicals for each abstract"

```
SM_AggregationOfLexicals(
    OID: #aggregationOID_1(OID),
    Name: n)
←
SM_Abstract (
    OID: OID,
    Name: n) ;
```

- the value for the attribute Name is copied (by using variable n)
- the value for OID is "invented": a new value for the function #aggregationOID_1(OID) for each different value of OID, so a different value for each value of SM_Abstract.OID
- Skolem functions are materialized in the dictionary:
  - **Represent the mapping**

"An aggregation of lexicals for each abstract"



SM-Abstract		
OID	**Schema**	**Name**
301	1	Employees
302	1	Departments
…	…	…

SM-AggregationOfLexicals		
OID	**Schema**	**Name**
1001	11	Employees
1002	11	Departments

SM-aggregationOID_1_SK	
OID	**absOID**
1001	301
1002	302

OID	Schema	N…	…OID
401	1	E…	1
402	1	N…	1
…	…		

Employees

```
SM_AggregationOfLexicals(
    OID: #aggregationOID_1(OID),
    Name: n)
←
SM_Abstract (
    OID: OID,
    Name: n) ;
```

"A component of the aggregation for each attribute of abstract"

```
SM_ComponentOfAggregation… (
    OID: #componentOID_1(attOID),
    Name: name,
    AggrOID: #aggregationOID_1(absOID),
    IsNullable: isNullable,
    IsKey: isIdent,
    Type : type )
←
SM_AttributeOfAbstract(
    OID: attOID,
    Name: name,
    AbstractOID: absOID,
    IsIdent: isIdent,
    IsNullable: isNullable ,
    Type : type ) ;
```

- Skolem functions
  - are functions
  - are injective
  - have disjoint ranges
- the first function "generates" a new value
- the second "reuses" the value generated by the first rule

aggregation for each ...of abstract"

```
SM_ComponentOfAggregation… (
    OID: #componentOID_1(attOID),
    Name: name,
    AggrOID: #aggregationOID_1(absOID),
    IsNullable: isNullable,
    IsKey: isIdent,
    Type : type )
←
SM_AttributeOfAbstract(
    OID: attOID,
    Name: name,
    AbstractOID: absOID,
    IsIdent: isIdent,
    IsNullable: isNullable ,
    Type : type ) ;
```

Employees		
EmpNo	Name	

SM-AttributeOfAbstract						
OID	Schema	Name	isIdent	isNullable	Type	AbstrOID
401	1	EmpNo	T	F	Int	301
402	1	Name	F	F	Text	301
…	…	…	…	…	…	…

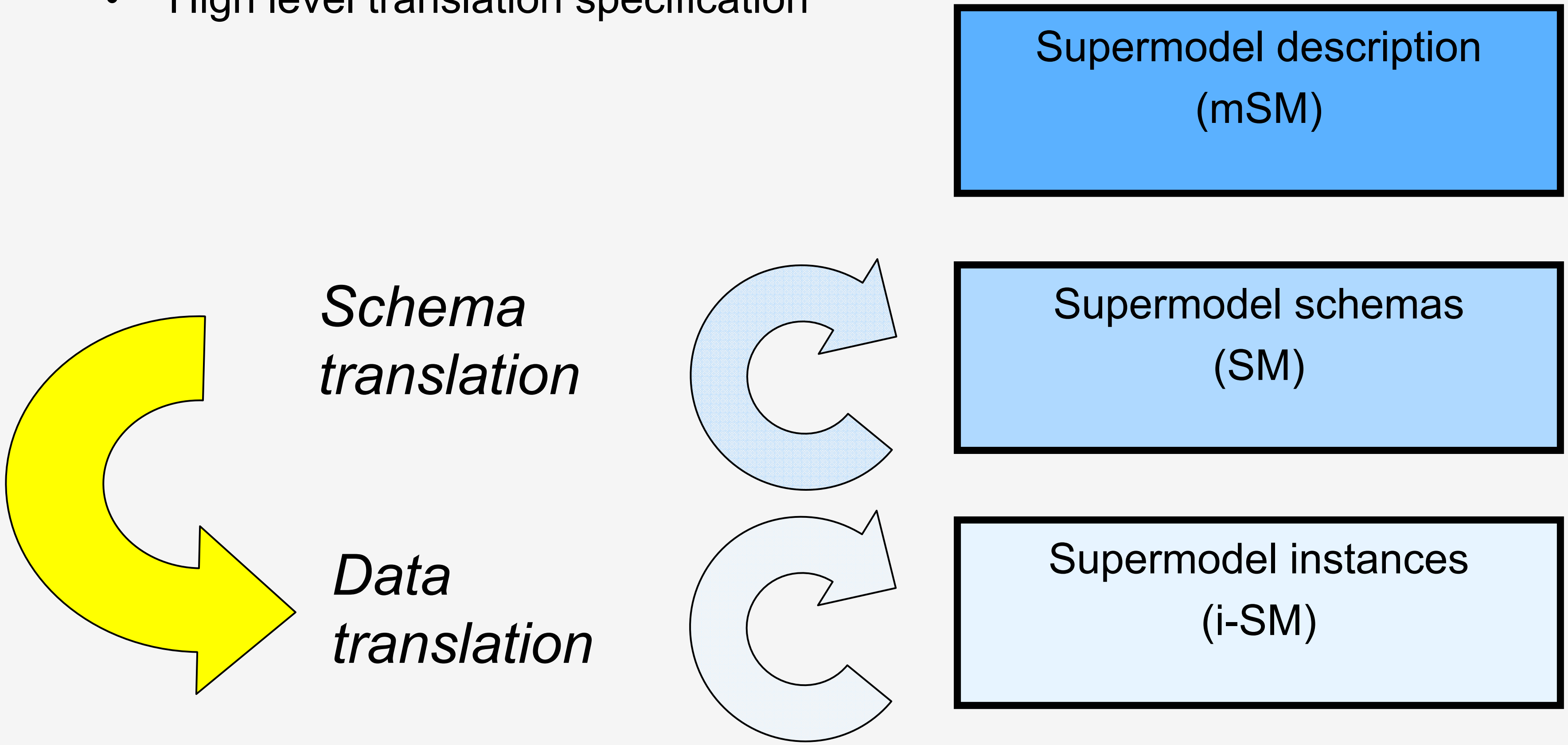
SM-ComponentOfAggregationOfLexicals						
OID	Schema	Name	isIdent	isNullable	Type	AggrOID
1003	11	EmpNo	T	F	Int	1001
1004	11	Name	F	F	Text	1001

SM-aggregationOID_1_SK	
OID	absOID
1001	301
1002	302

SM-componentOID_1_SK	
OID	absOID
1003	401
1004	402

Generating data-level translations

- Same environment
- Same language
- High level translation specification



Translation rules, data level

```
SM_ComponentOfAggregation… (
  OID: #componentOID_1(attOID),
  Name: name,
  AggrOID:
       #aggregationOID_1(absOID),
  IsNullable: isNullable,
  IsKey: isIdent,
  Type : type )
←
SM_AttributeOfAbstract(
  OID: attOID,
  Name: name,
  AbstractOID: absOID,
  IsIdent: isIdent,
  IsNullable: isNullable ,
  Type: type ) ;
```

```
i-SM_ ComponentOfAggregation… (
    OID: #i-componentOID_1 (i-attOID),
    i-AggrOID: #i-aggregationOID_1(i-absOID),
    ComponentOfAggregationOfLexicalsOID:
      #componentOID_1(attOID),
    Value: Value )
←
i-SM_AttributeOfAbstract(
    OID: i-attOID,
    i-AbstractOID: i-absOID,
    AttributeOfAbstractOID: attOID,
    Value: Value ),
SM_AttributeOfAbstract(
    OID: attOID,
    AbstractOID: absOID,
    Name: attName,
    IsNullable: isNull,
    IsID: isIdent,
    Type: type )
```

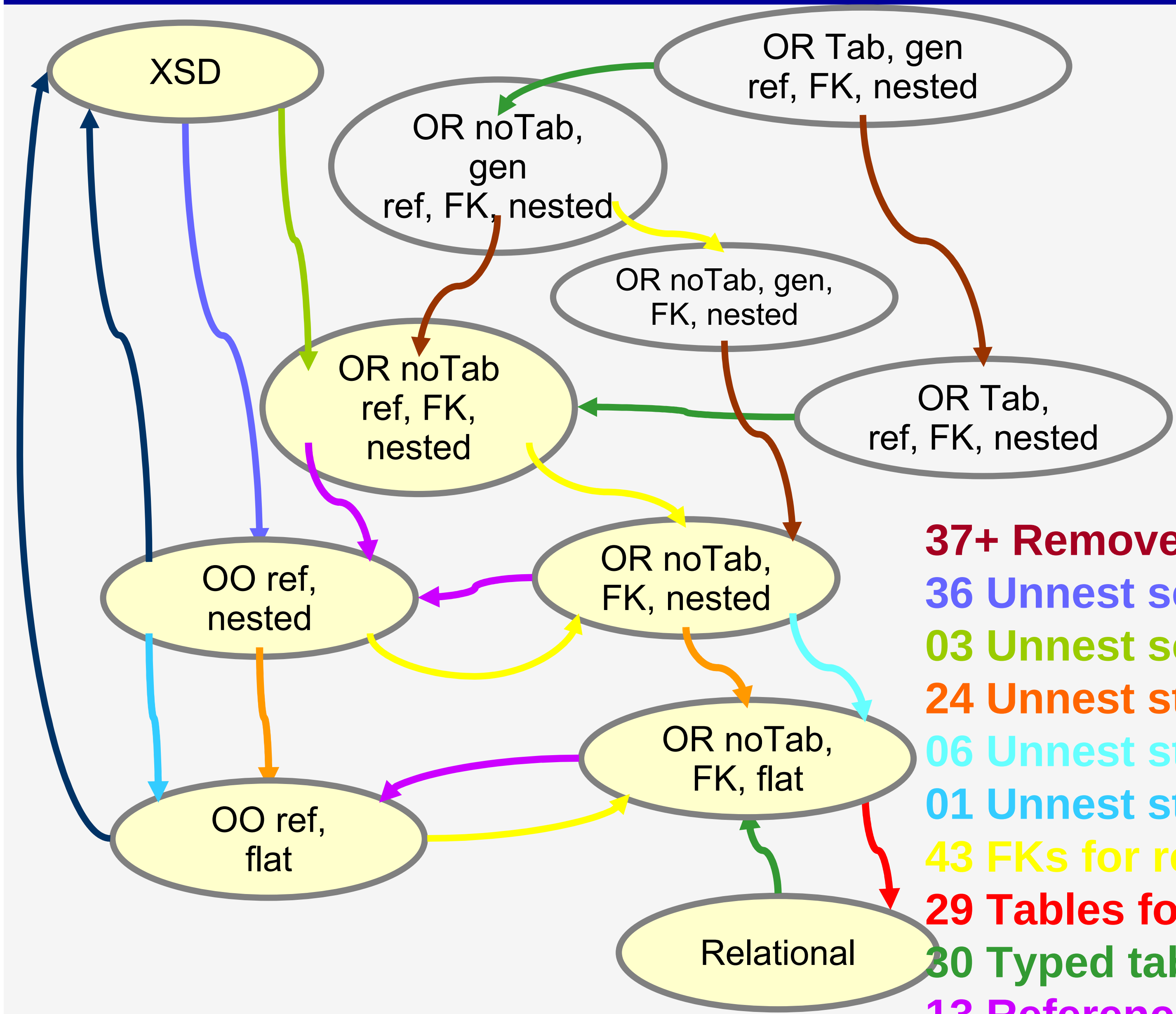
Correctness

- Usually modelled in terms of information capacity equivalence/dominance (Hull 1986, Miller 1993, 1994)
- Mainly negative results in practical settings that are non-trivial
- Probably hopeless to have correctness in general
- We follow an "axiomatic" approach:
  - We have to verify the correctness of the basic translations, and then infer that of complex ones

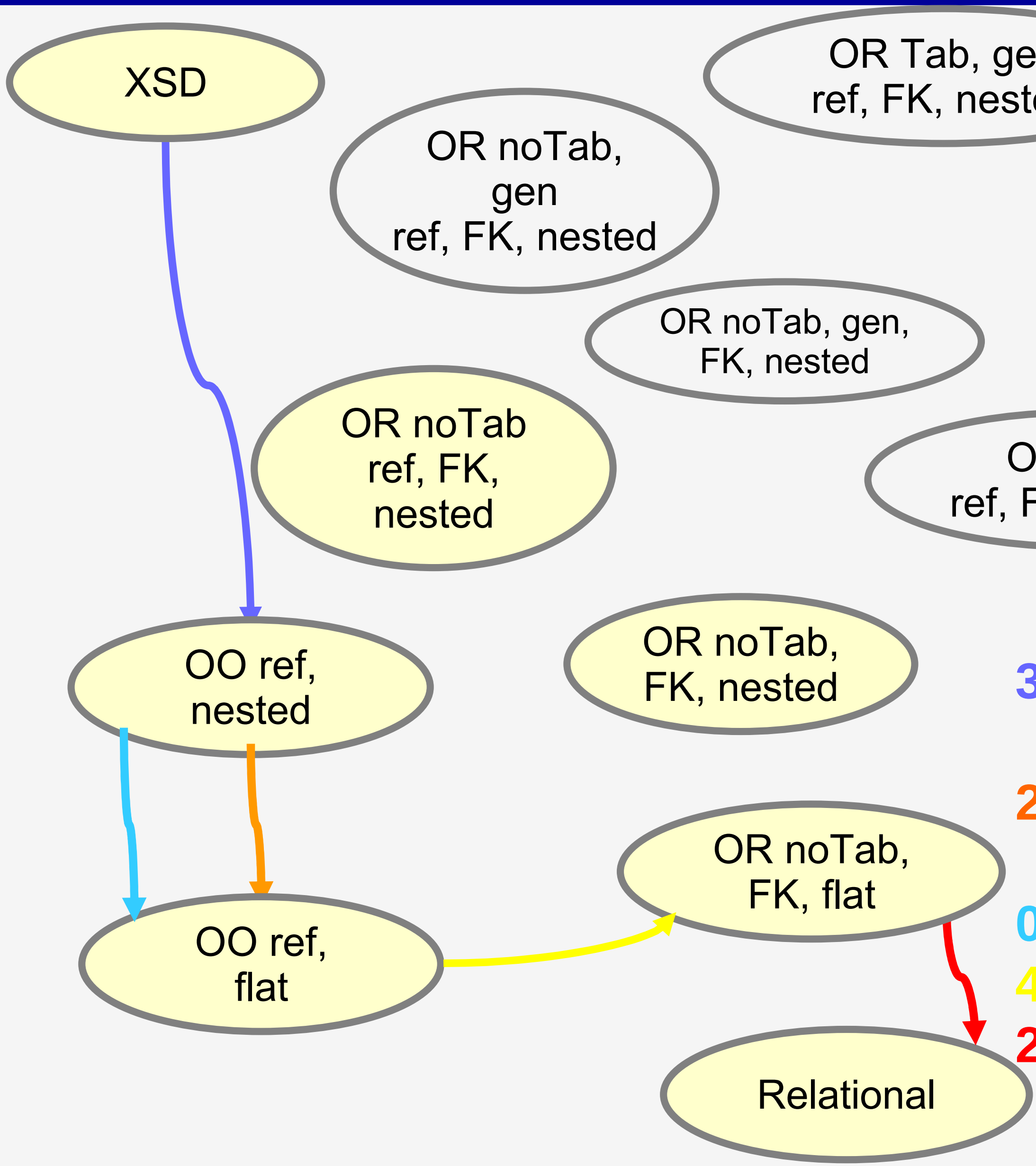
Experiments

- A significant set of models
  - ER (in many variants and extensions)
  - Relational
  - OR
  - XSD
  - UML

- **Demo**



37+ Remove generalizations
36 Unnest sets (with ref)
03 Unnest sets (with FKs)
24 Unnest structures (flatten)
06 Unnest structures (TT & FKs)
01 Unnest structures (TT & ref)
43 FKs for references
29 Tables for typed tables
30 Typed tables for tables
13 References for FK
31 Nest referenced classes



36 Unnest sets (with ref)

24 Unnest structures (flatten)

01 Unnest structures (TT & ref)

43 FKs for references

29 Tables for typed tables

Summary

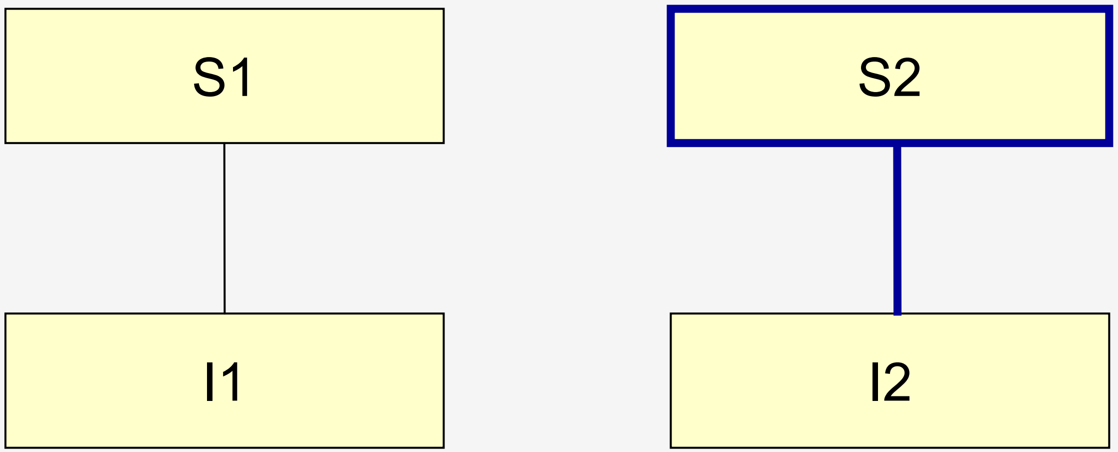
- ModelGen was studied a few years ago
- New interest on it within the "Model management" framework
- New approach
  - Translation of schema and data
  - Visible (and in part self generated) dictionary
  - Visible and modifiable rules
  - Skolem functions describe mappings

Issues

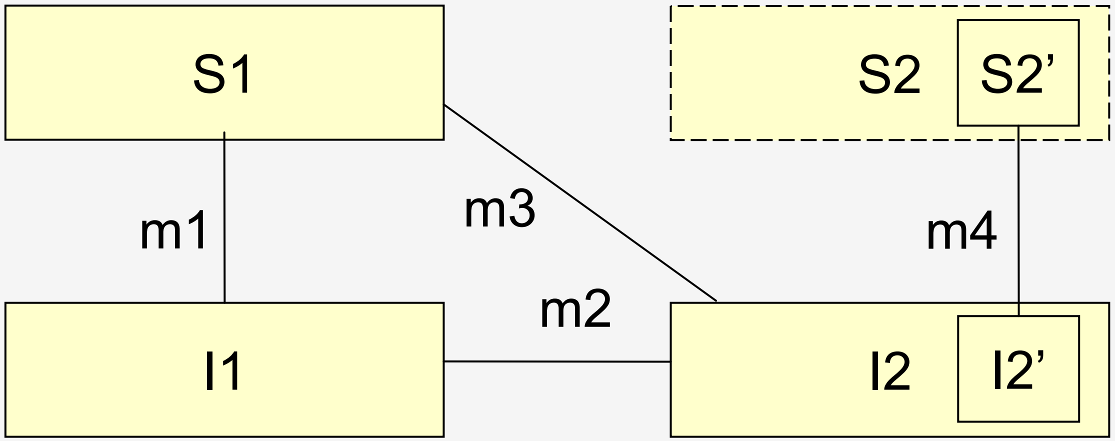
- ModelGen in the model management scenario:
  - Round-trip engineering (and more)
- Customization of translations
- Off-line translation vs run-time
- “Correctness” of data translation wrt schema translation:
  - compare with data exchange
- Schematic heterogeneity and semantic Web framework:
  - What if the distinction between schemas and instances is blurred?
- Materialized Skolem function & provenance

Modelgen in model management

- Round trip engineering (Bernstein, CIDR 2003)
  - A specification (for example ER or UML) and an implementation (for example, relational)
  - then a change to the implementation: want to revise the specification
- We need a translation from the implementation model to the specification one



Round trip engineering



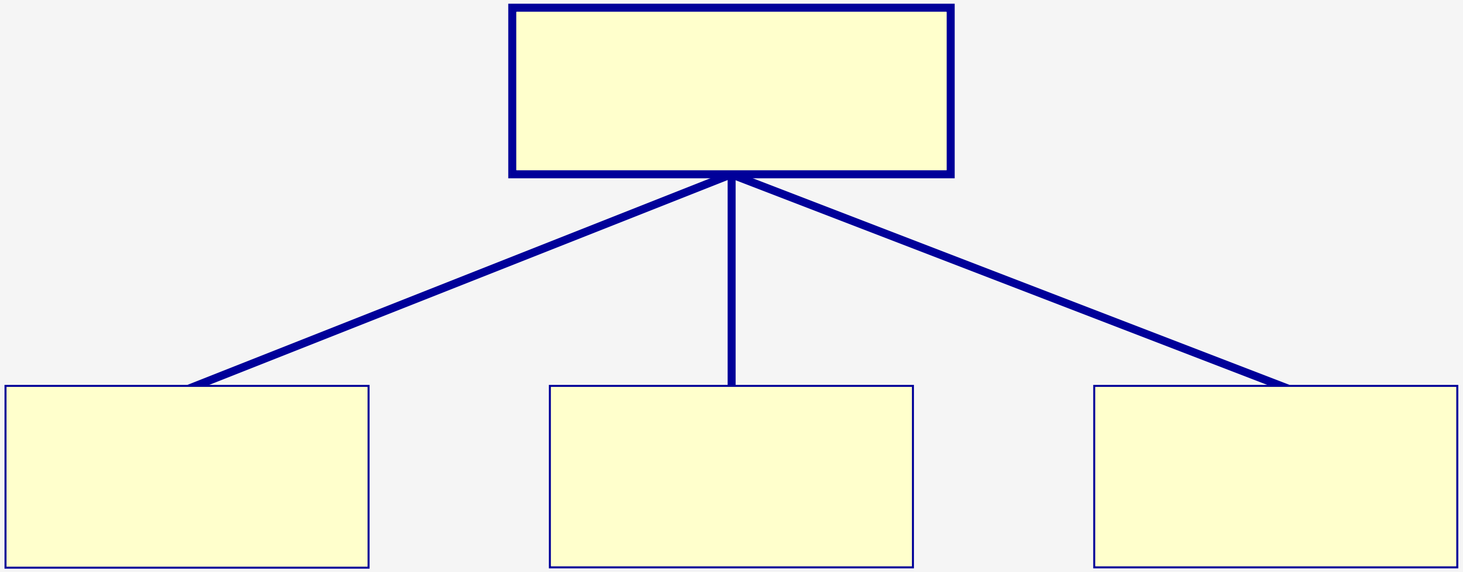
m2 = Match (I1,I2)
m3 = Compose (m1,m2)
I2'= Diff(I2,m3)
<S2',m4 > = Modelgen(I2')
… Match, Merge

Another problem in the picture: data exchange

- Given a source S1 and a target schema S2 (in different models or even in the same one), find a translation, that is, a function that given a database D1 for S1 produces a database D2 for S2 that “correspond” to D1
- Often emphasized with reference to materialized solutions

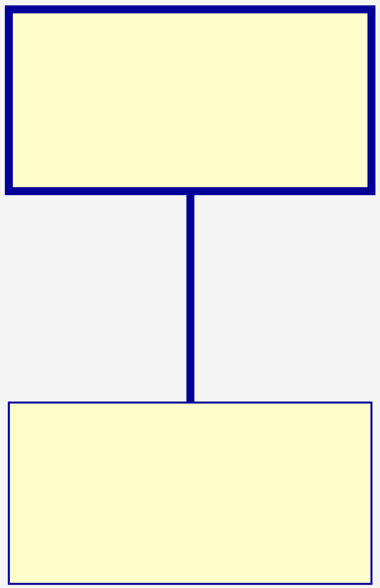
Integration

- Given two or more sources, build an integrated schema or database



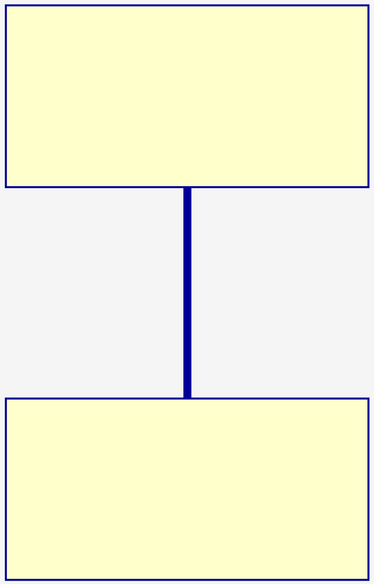
Schema translation

- Given a schema find another one with respect to some specific goal (**another model**, better quality, …)



Data exchange

- Given a source and a target schema, find a transformation from the former to the latter



Schema translation and data exchange

- Can be seen as complementary:
  - Data translation = schema translation + data exchange
    - Given a source schema and database
    - Schema translation produces the target schema
    - Data exchange generates the target database

- In model management terms we could write
  - Schema translation:
    - <S2, map12> = **ModelGen** (S1,mod2)
  - Data exchange:
    - i2 = **DataGen** (S1,i1,S2,map12)