# Using Model Management for Data Integration

Phil Bernstein
Microsoft Research

Nov. 26, 2007

Most slides come from SIGMOD 07 Keynote & "Bridging Apps & DB", both with Sergey Melnik

# Data Programmability

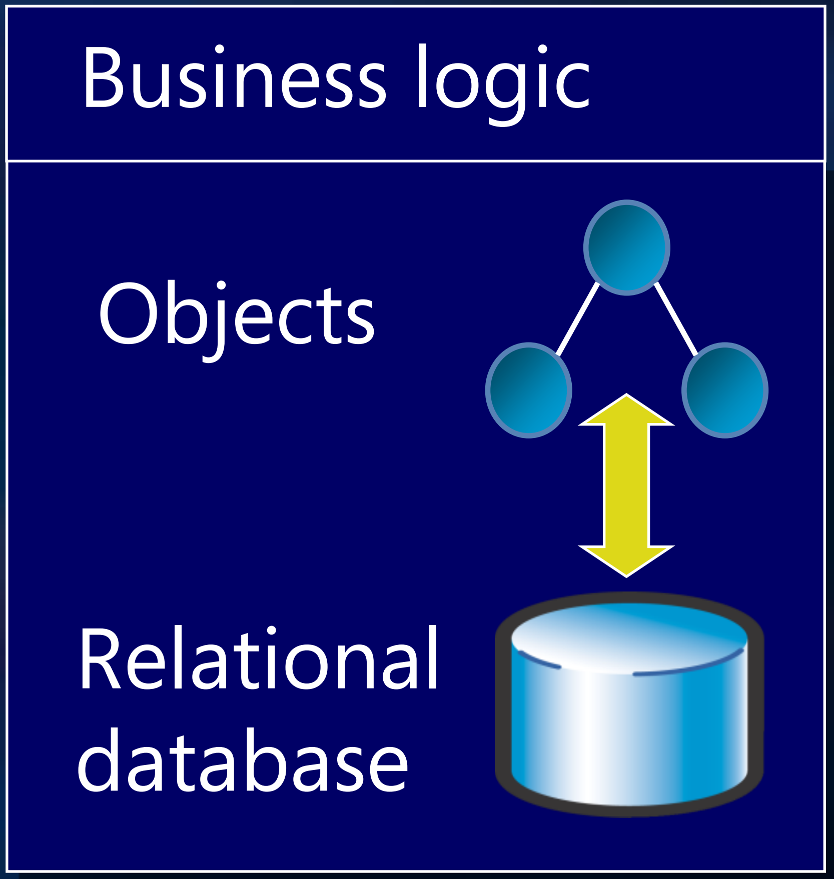
- Make it easier to write programs that access databases
- Traditionally, for large IT departments
- Much progress, but it's still ~40% of the work
- Core problem is developing and using complex mappings between schemas

# Mapping Problems are Pervasive
## And it's a Growth Industry

- Data translation
- XML message mapping
- Data warehouse loading
- Query mediators
- Forms managers
- Report writers
- Query designers
- Object-relational wrappers
- Portal generation from DB
- OLAP databases
- Application integration
- Composing web services

# Object-Relational Wrappers

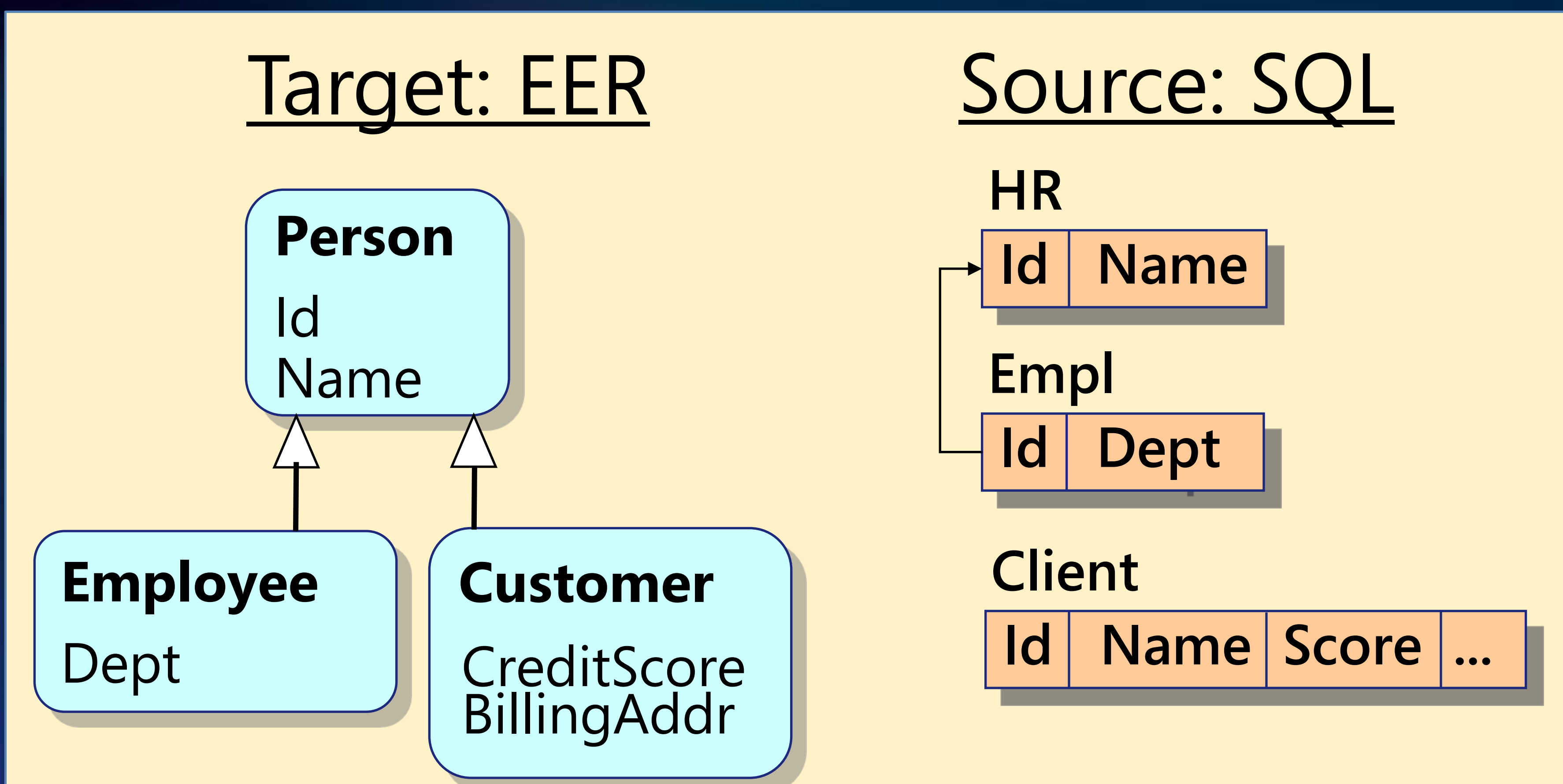
- Most packaged business apps need to access an OO view of relational data
- Requires an OR wrapper
- App developer specifies a high-level mapping
- A tool translates the mapping into executable code



# An Example Mapping

- Person = HR $\cup$ $\pi_{ID,Name}$(Client)
- Employee = HR $\bowtie$ Empl
- Customers = Client

*Specified by app developer*



# Executable Code for Persons

[ Melnik, Adya, Bernstein, SIGMOD 07]

```
SELECT VALUE
    CASE
      WHEN (T5._from2 AND NOT(T5._from1))   THEN Person(T5.Person_Id, T5.Person_Name)
      WHEN (T5._from1 AND T5._from2)
         THEN Employee(T5.Person_Id, T5.Person_Name,  T5.Employee_Dept)
         ELSE Customer(T5.Person_Id, T5.Person_Name,  T5.Customer_CreditScore,
              T5.Customer_BillingAddr)
    END
FROM (   (SELECT T1.Person_Id, T1.Person_Name,  T2.Employee_Dept,
                   CAST(NULL AS SqlServer.int) AS Customer_CreditScore,
                   CAST(NULL AS SqlServer.nvarchar) AS Customer_BillingAddr, False AS _from0,
                   (T2._from1 AND T2._from1 IS NOT NULL) AS _from1,  T1._from2
          FROM  ( SELECT  T.Id AS Person_Id, T.Name AS Person_Name, True AS _from2
                    FROM HR AS T) AS T1
                    LEFT OUTER JOIN (
                      SELECT  T.Id AS Person_Id,  T.Dept AS Employee_Dept, True AS _from1
                      FROM dbo.Empl AS T) AS T2
                    ON T1.Person_Id = T2.Person_Id )
          UNION ALL (
                  SELECT  T.Id AS Person_Id,  T.Name AS Person_Name,
                          CAST(NULL AS SqlServer.nvarchar) AS Employee_Dept,
                          T.Score AS Customer_CreditScore,  T.Addr AS Customer_BillingAddr,
                          True AS _from0,  False AS _from1,  False AS _from2
                  FROM Client AS T)
        ) AS T5
```

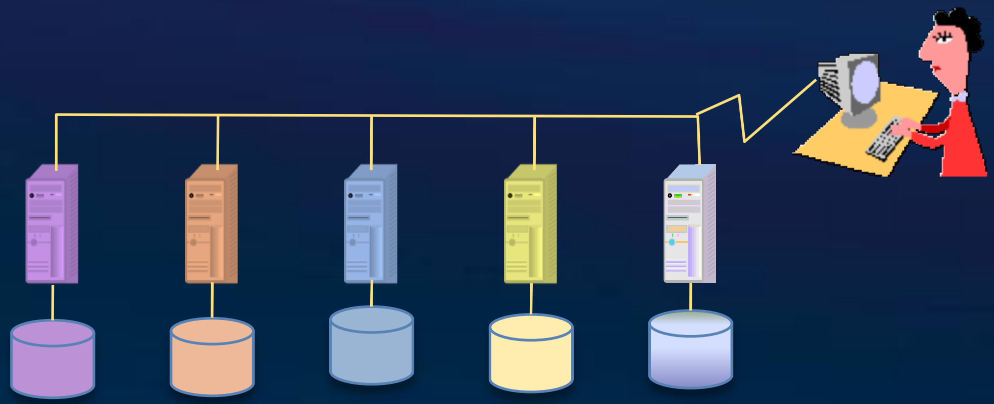
# The Theme

- The main benefit
  - It’s easier to design mappings than to write code
- The main problems
  - Help the user develop mappings
  - Translate mappings into code

# Why is mapping hard?

[ Haas, ICDT 07]

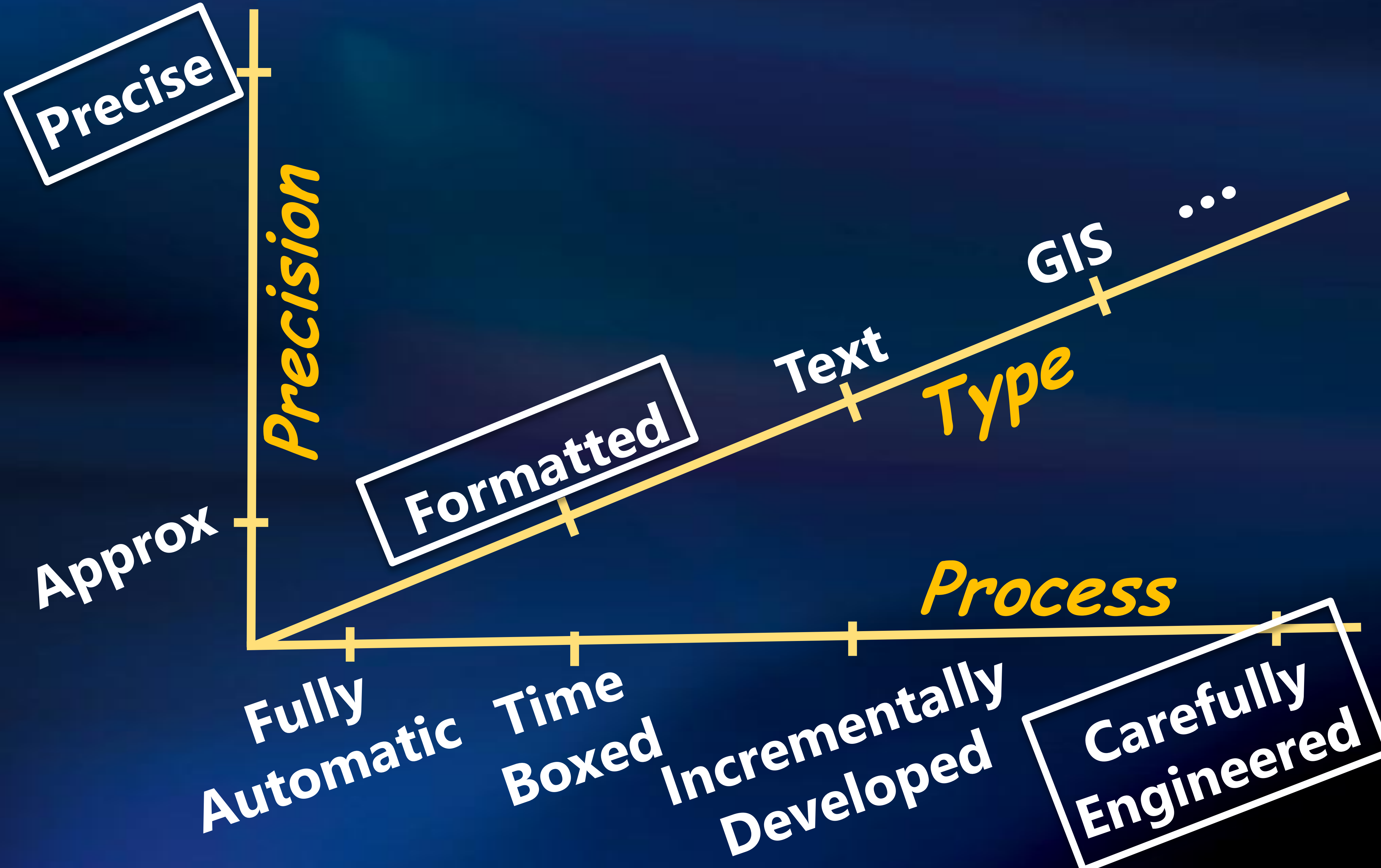
- Heterogeneity
- Impedance mismatch
- Insufficient abstraction
- Potpourri of tools



- And it's getting harder due to more
  - Programming languages
  - Types of tools
  - Data models
  - Schema sources

# Mapping Space

Info Integration Workshop
http://db.cis.upenn.edu/iiworkshop/



# Outline

- ✓ Motivation
- Model Management
- Operators & Scenarios

# Model Management

[ Bernstein, Halevy, Pottinger SIGMOD Record 00]

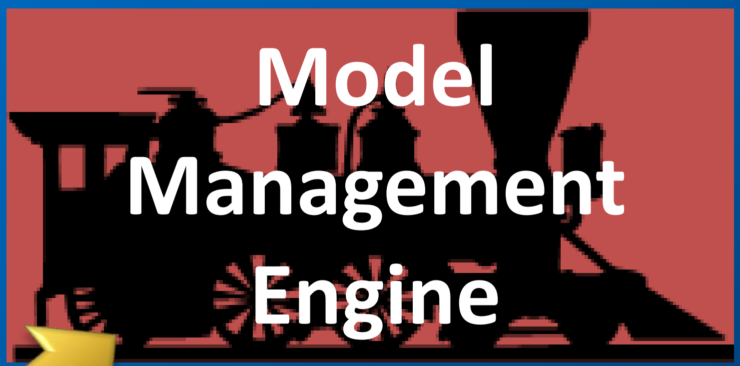
**Manipulate**
***models & mappings***
**as bulk objects**

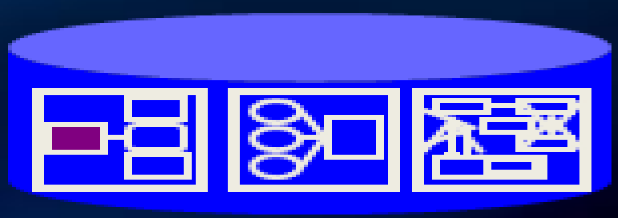

**Meta-model independent**
- relational, ER, OO, XML, RDF, OWL, SML, …

**Operations**
- Match
- Compose
- Merge
- Diff/Extract
- ModelGen
- Inverse

Tools

- Wrapper Generator
- Query Mediator
- ETL



Metadata Repository

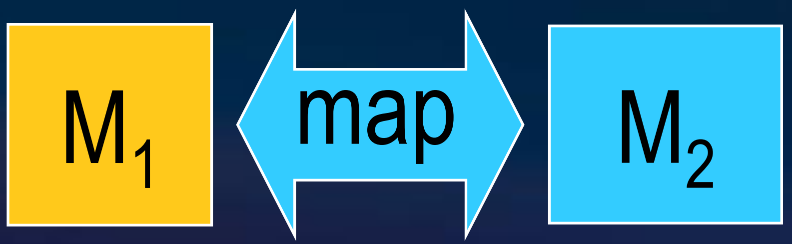
# Model Management
## Getting Started

- Choose a schema definition language
- Choose a mapping language

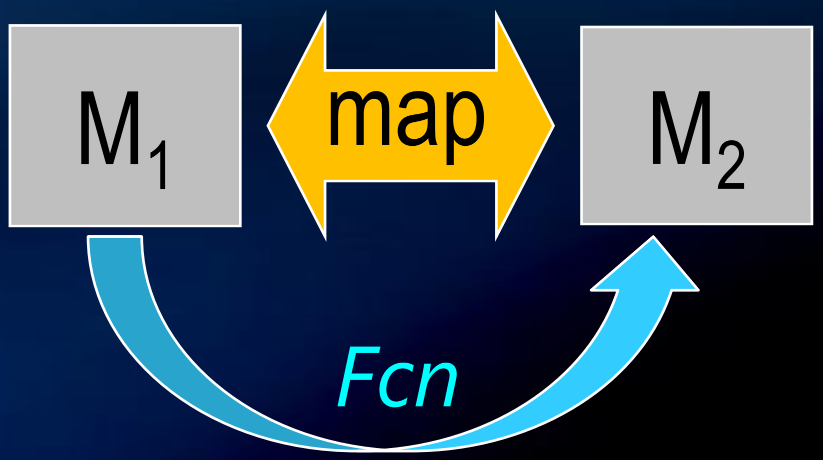
# Model Mgmt Operators

*map* = Match($M_1$, $M_2$)


⟨$M_2$ , *map*⟩ =
ModelGen($M_1$, $metamodel_2$)



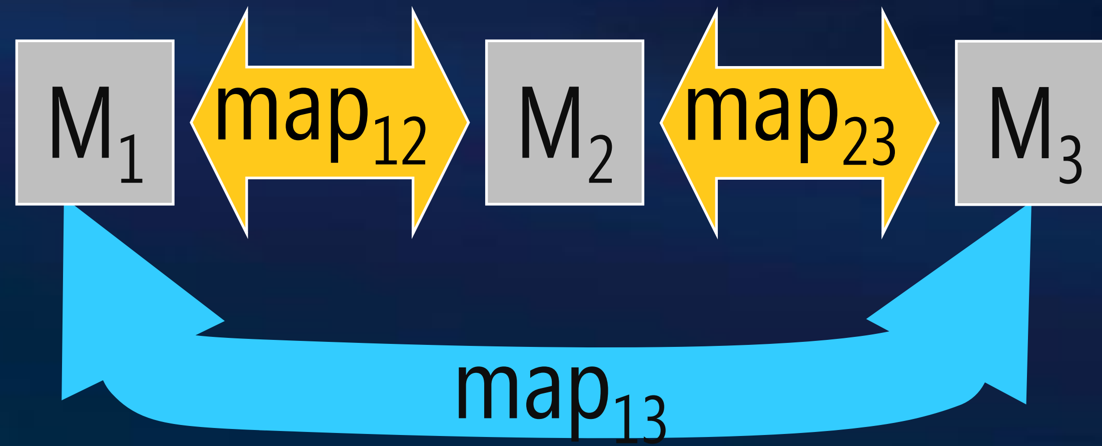

*Fcn* = TransGen(*map*)



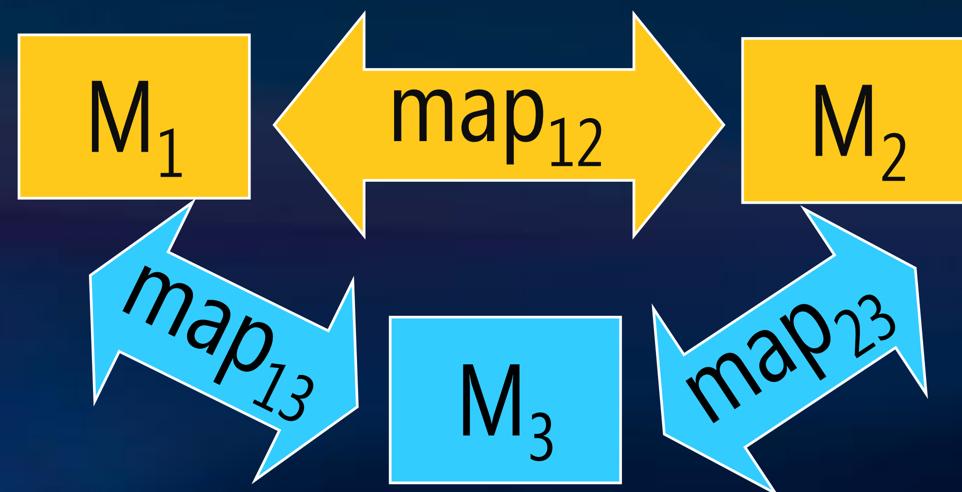
# Model Mgmt Operators (cont'd)

Compose:

$map_{13} = map_{12} \bullet map_{23}$




$\langle M_3 , map_{13} , map_{23} \rangle =$ Merge $(M_1 , M_2 , map_{12})$




$\langle M_2 , map_2 \rangle =$ Diff$(M_1 , map_1 )$


Plus a few more …

# Model Management Benefits

- More research focus on primary operations
  - More powerful operations
  - Hence better tools
- More leverage from tool investments
- More uniform behavior across tools

# Status Report

- Good News
  - Lots of progress on operations
  - Some practical applications
  - A lot has been learned
- Bad News
  - Still waiting for the first reasonably-complete practical implementation
- Good news
  - A lot of research left to do

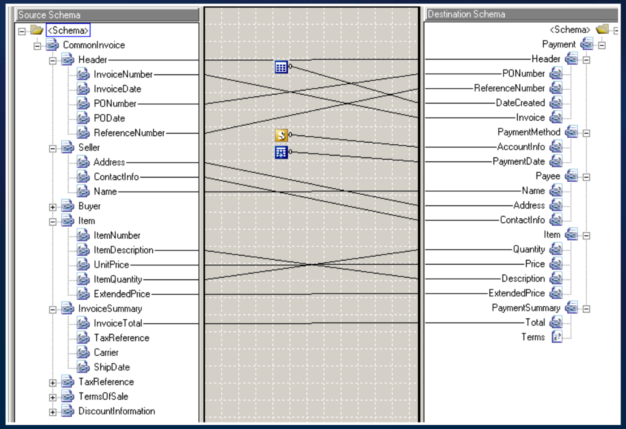
# Semantic Mapping

- $\mathbb{I}(S_1)$ are the instances of schema $S_1$
  - Each d in $\mathbb{I}(S_1)$ is a database (e.g., a set of relations)
- $\mathbb{I}(S_2)$ are the instances of schema $S_2$
- $map_{12} \subseteq \mathbb{I}(S_1) \times \mathbb{I}(S_2)$
- Usually, we represent a mapping by an expression
  - $V = R \bowtie S$
  - $R \bowtie S = T \bowtie U$

# Mappings

[Casanova, Vidal. PODS 83] [Catarci, Lenzerini. J. CoopIS 93]
[Biskup, Convent. SIGMOD 86] [Miller, Haas, Hernandez. VLDB 00]

- Element correspondences
  - First step in aligning schemas
  - For lineage & impact analysis
  - Weak or no formal semantics




- Mapping constraints relate instances of schemas
  - E.g., equality of relational expressions

```
SELECT Id, Name, Dept  =  SELECT Id, Name, Dept
FROM   Employee            FROM HR JOIN Empl ON Id
```

- Transformation is an executable mapping constraint
  - Constructs target instances from source instances
  - E.g., SQL query, XSLT, C# program

# Mapping Expressiveness

- What we want: first-order logic with
  - negation
  - aggregation
  - set and bag semantics
  - regular expressions
  - nested collections and lists
  - rich type constructors (e.g., to construct XML fragments),
  - user-defined functions
  - deduplication and other heuristic functions
- What can we handle? … Much less.

# Outline

- ✓ Motivation
- ✓ Model Management
- Operators & Scenarios

# Scenarios

1. **Create mappings**
   - ModelGen
   - Match
   - ConstraintGen
   - TransGen

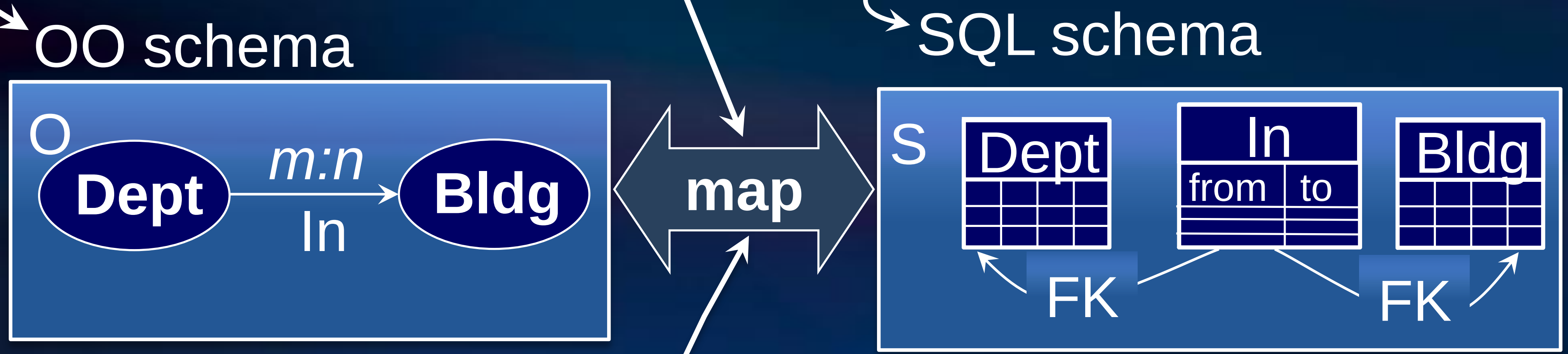
2. **Evolve mappings**
   - Compose
   - Diff
   - Merge
   - Inverse

# ModelGen: Schema Translation

Input
- source model
- target metamodel

Output
- target model
- constraints

[Atzeni, Torlone. EDBT 96]
[Bernstein, Melnik, Mork. VLDB 05]
[Atzeni, Cappellari, Bernstein. EDBT 06]

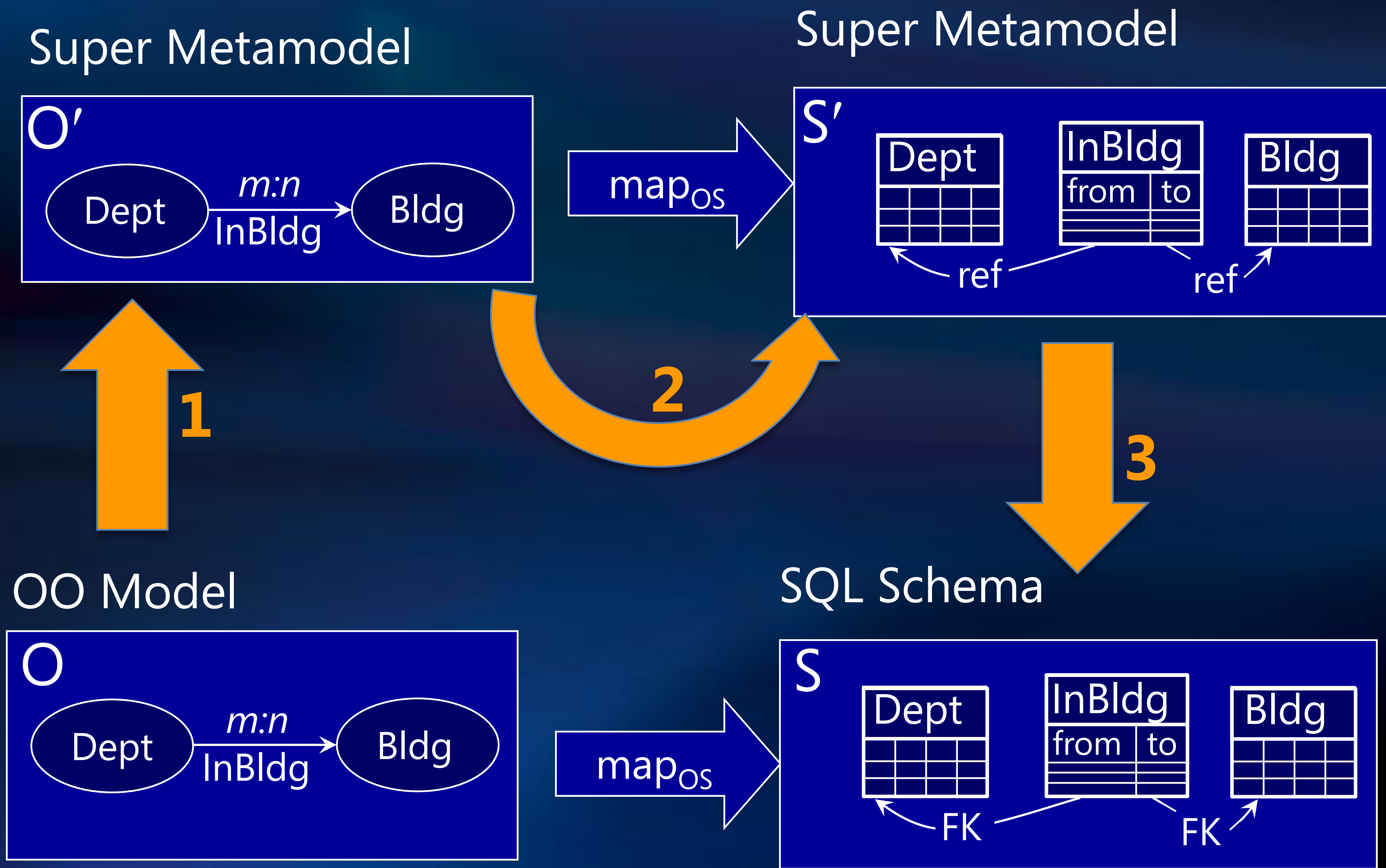



O.Dept(d) ⇔ S.Dept(d.key)
O.Bldg(b) ⇔ S.Bldg(b.key)
O.In(d,b) ⇔ S.In(d.key, b.key)

- There are several credible prototypes
  - Don't know of products, yet

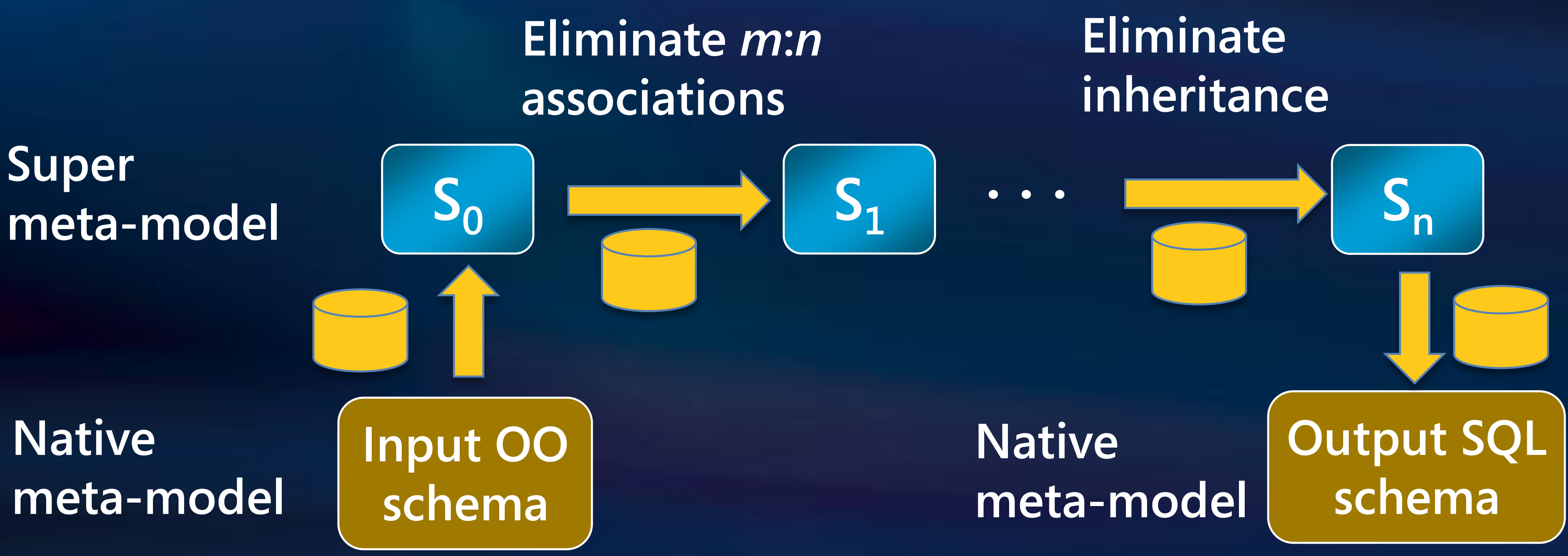
# Implementation Strategy

[Atzeni & Torlone, EDBT '96]



# Moving Data via ModelGen

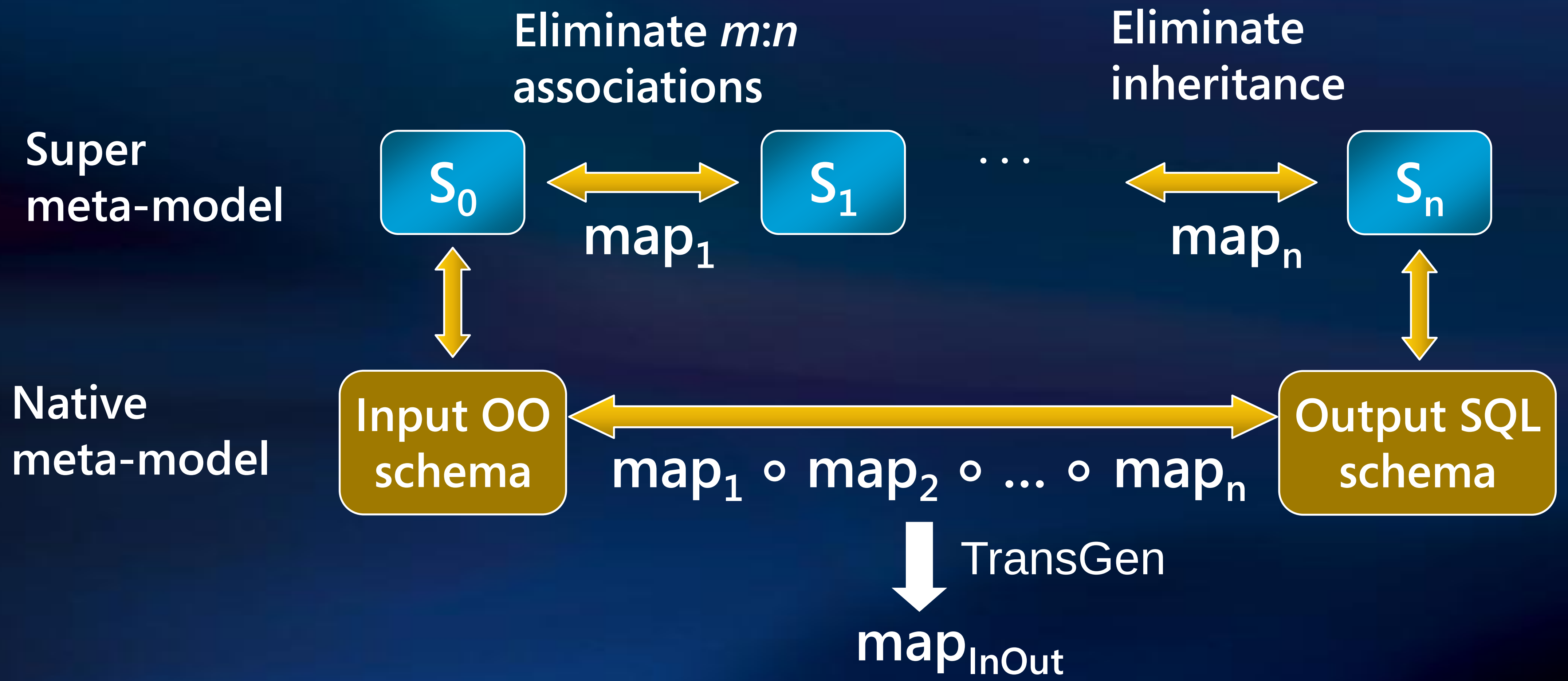
[Papotti, Torlone]
[Atzeni, Cappellari]




- Data is transferred to super-metamodel DB
- Data is transformed within super-metamodel DB
- Data is transferred to output schema's database

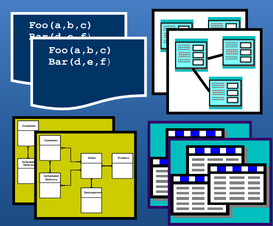
# Obtaining Mappings From ModelGen

[Bernstein, Melnik, Mork VLDB'05, ER'07]



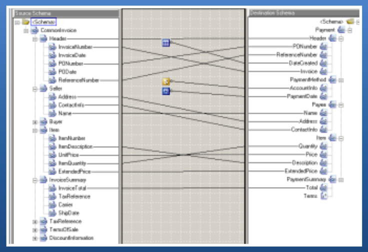

- Leverages Compose operator
- Each $map_i$ roundtrips data

# Code Generation Scenarios [Miller, Haas, & Hernandez, VLDB 00]



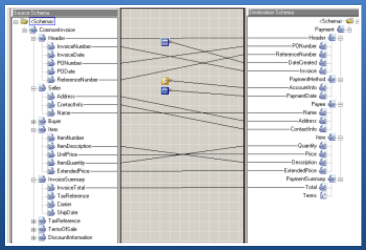
Models

**Match** →



Correspondences

**ModelGen** →



Correspondences

**ConstraintGen** →

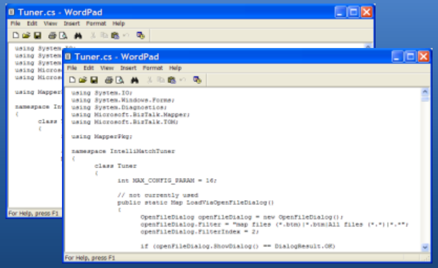
Select ord#, prod#, cust#
From Shipped
=
Select ord#, prod#, cust#
From Order Join Item
on ord#

Constraints

Select ord#, prod#, cust#
From Shipped
⊆
Select ord#, prod#, cust#
From Order Join Item
on ord#

Constraints

**TransGen** →



Transformations

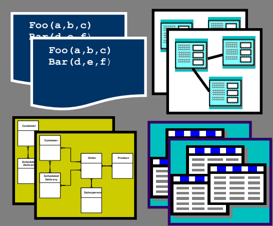
# Schema Matching


- Produce candidate correspondences
- Exploit lexical analysis of element names, schema structure, data types, thesauri, value distributions, ontologies, instances, and previous matches
- Past Goal - improved precision & recall
  - Big productivity gains are unlikely
- Better goals
  - Return top-k, not best overall match
  - Avoid the tedium. Manage work.
  - HCI – handle large schemas.
  - User studies – what would improve productivity?

# Cast of Thousands

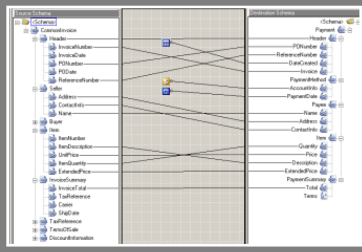
- AnHai Doan
- Alon Halevy
- Pedro Domingos
- Phil Bernstein
- Erhard Rahm
- Sergey Melnik
- Jayant Madhavan
- Jeffrey Naughton
- Jaewoo Kang
- Tova Milo
- Pavel Shvaiko
- Fausto Giunchiglia
- Sonia Bergamaschi
- Silvana Castano
- Bin He
- Kevin Chang
- Namyoun Choi
- Il-Yeol Song
- Hyoil Han
- Domenico Ursino
- Luigi Palopoli
- Dominico Sacca
- Georgio Terracina
- David Embley
- David Jackman
- Li Xu
- Yihong Ding
- Jacob Berlin
- Amihai Motro
- Hong Hai Do
- Fabien Duchateau
- Zohra Bellahsene
- Ela Hunt
- Toralf Kirsten
- Andreas Thor
- Alexander Bilke
- Avigdor Gal
- Michalis Petropoulos
- Christoph Quix
- Chris Clifton
- Arnie Rosenthal
- Wen-Syan Li
- Hector Garcia-Molina
- Sagit Zohar
- Gio Wiederhold
- Anna Zhdanova
- Jerome Euzenat
- Prasenjit Mitra
- Natasha Noy
- Anuj Jaiswal
- Mikalai Yatskevich
- Nuno Silva
- Joao Rocha
- David Aumueller
- Sabine Massmann
- Felix Naumann

# Code Generation Scenarios [Miller, Haas, & Hernandez, VLDB 00]

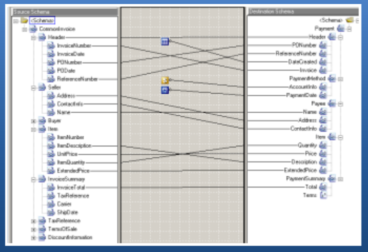


Models

**Match**



Correspondences



Correspondences

**ConstraintGen**

Select ord#, prod#, cust#
From Shipped
=
Select ord#, prod#, cust#
From Order Join Item
on ord#

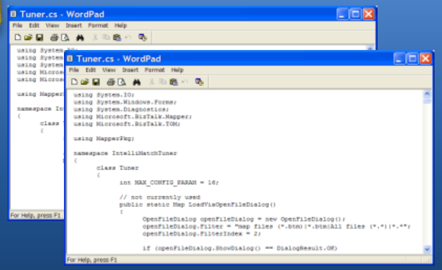
Constraints

**?**

Select ord#, prod#, cust#
From Shipped
⊆
Select ord#, prod#, cust#
From Order Join Item
on ord#

Constraints

**TransGen**



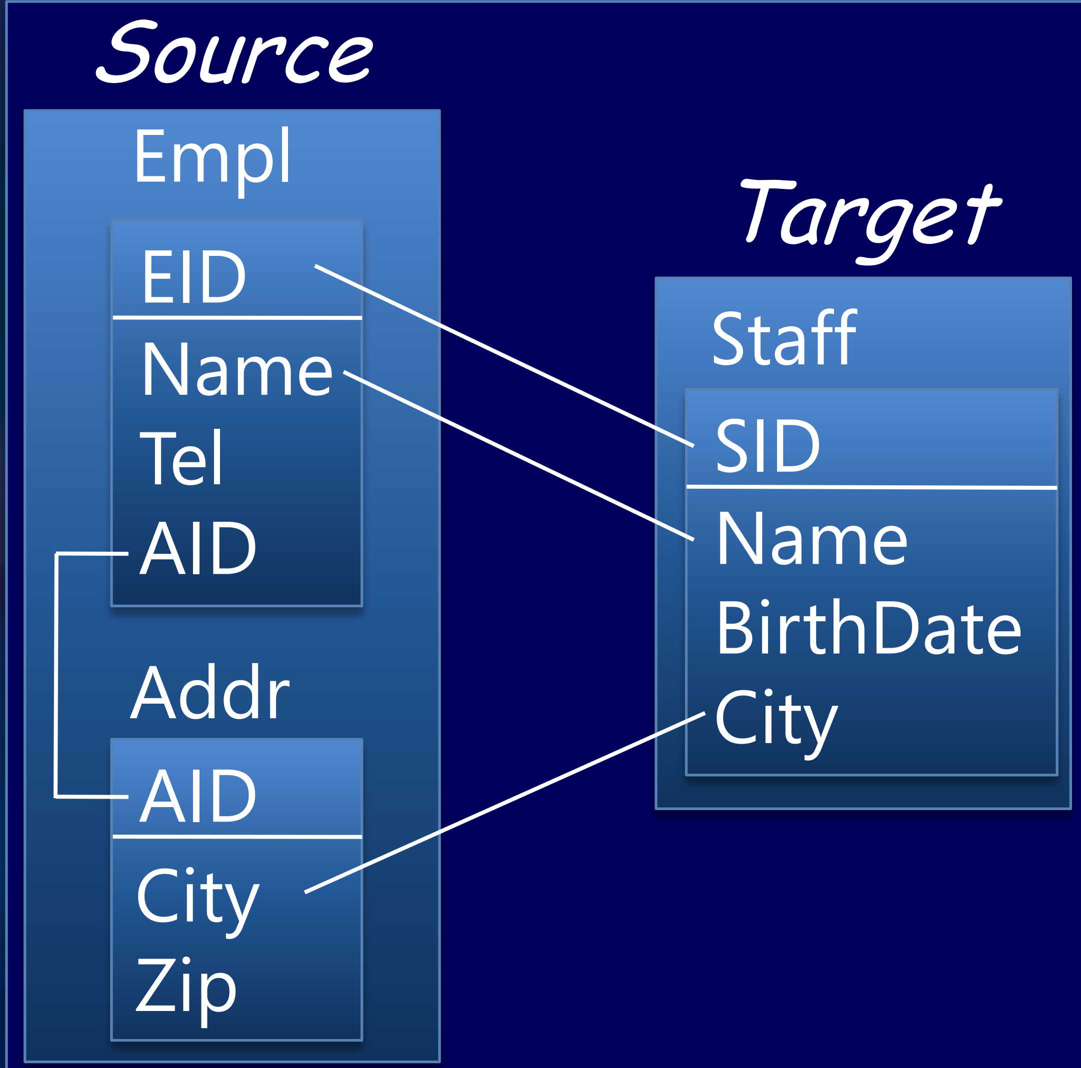
Transformations

# Correspondences → Transformations

[Popa, Velegrakis, Miller, Hernandez, Fagin. VLDB 02]
[Velegrakis. PhD thesis 2005]

For a given target element

- Find all source elements linked by correspondences
- Find all ways that source elements are related
- Choose one of them and generate the transformation

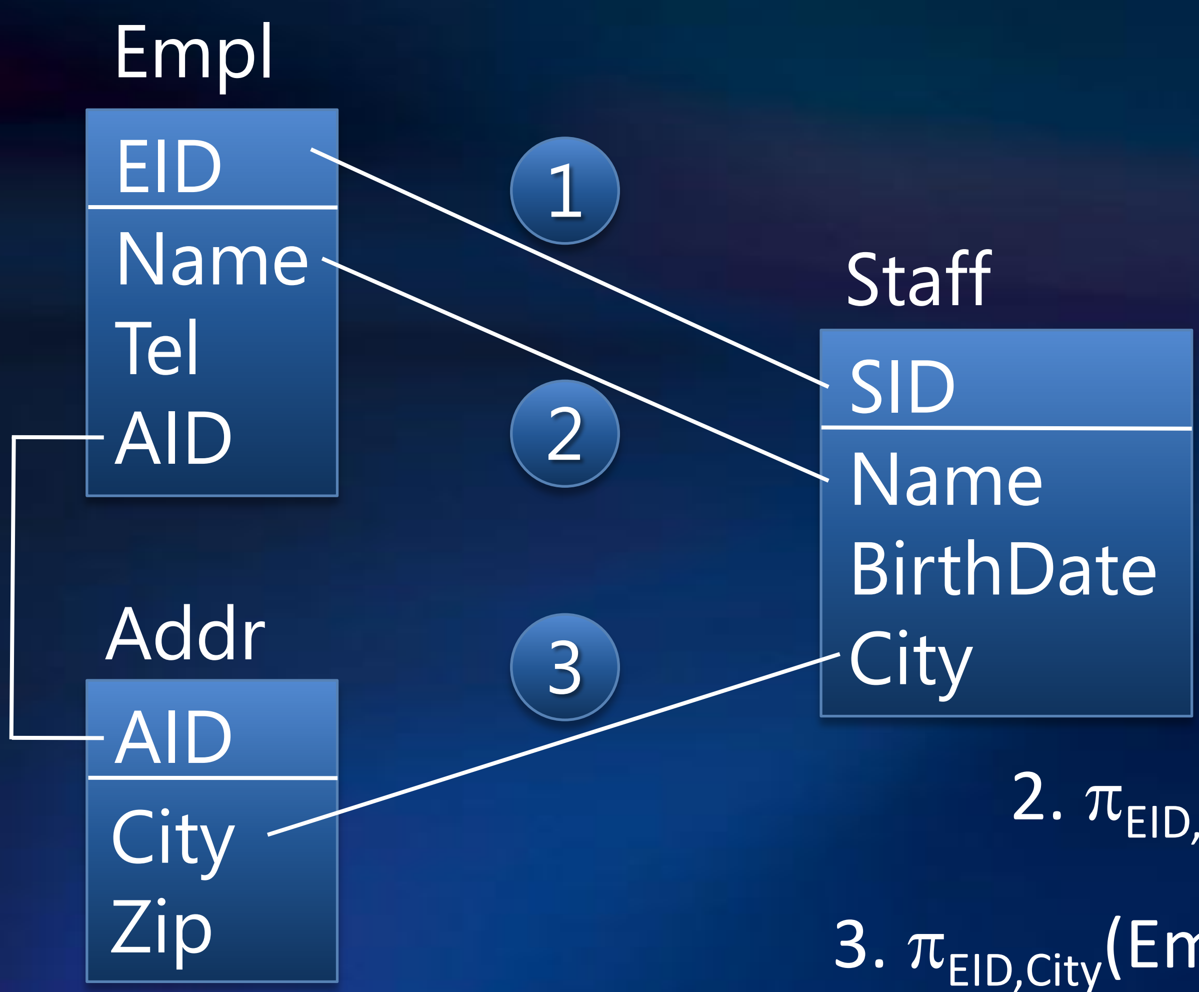



$$\pi_{SID,Name,City}(Staff) = \pi_{EID,Name,City}(Empl \bowtie Addr)$$

# Correspondences → Constraints

[Melnik, Bernstein, Halevy, & Rahm, SIGMOD 05]

- Directly interpret correspondences as mapping constraints
- If it's a tree schema and keys correspond

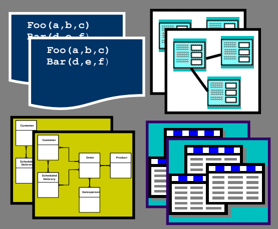



1. $\pi_{EID}(Empl) = \pi_{SID}(Staff)$

2. $\pi_{EID,Name}(Empl) = \pi_{SID,Name}(Staff)$

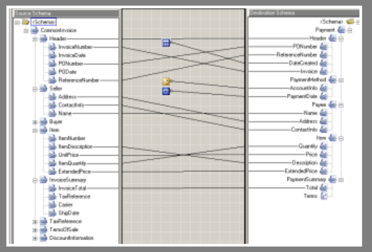
3. $\pi_{EID,City}(Empl \bowtie Addr) = \pi_{SID,City}(Staff)$

# Code Generation Scenarios

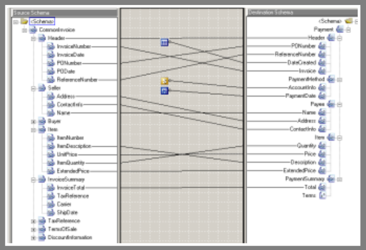



Models

**Match**



Correspondences



Correspondences

**ConstraintGen**

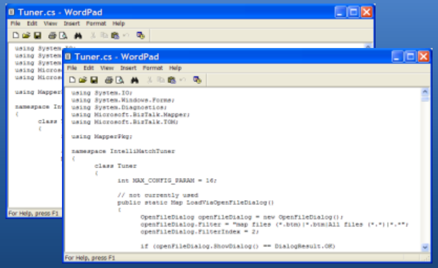
Select ord#, prod#, cust#
From Shipped
=
Select ord#, prod#, cust#
From Order Join Item
on ord#

Constraints

Select ord#, prod#, cust#
From Shipped
⊆
Select ord#, prod#, cust#
From Order Join Item
on ord#

Constraints

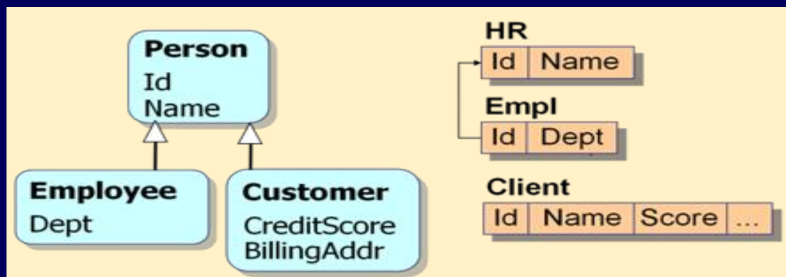
**TransGen**



Transformations

# Constraints → Transformations

[ Melnik, Adya, Bernstein, SIGMOD 07]




```
SELECT p.Id, p.Name
FROM Persons AS p
WHERE p IS OF (ONLY Person)
  OR p IS OF (ONLY Employee)
=
SELECT Id, Name
FROM dbo.HR
```

```
SELECT e.Id, e.Dept
FROM Persons AS e
WHERE e IS OF Employee
=
SELECT Id, Dept
FROM dbo.Empl
```

```
SELECT c.Id, c.Name,
   c.CreditScore, c.BillingAddr
FROM Persons AS c
WHERE c IS OF Customer
=
SELECT Id, Name,
         Score, Addr
FROM dbo.Client
```

```
SELECT VALUE
   CASE
     WHEN (T5._from2 AND NOT(T5._from1))   THEN Person(T5.Person_Id,
T5.Person_Name)
     WHEN (T5._from1 AND T5._from2)
       THEN Employee(T5.Person_Id, T5.Person_Name,  T5.Employee_Dept)
       ELSE Customer(T5.Person_Id, T5.Person_Name,  T5.Customer_CreditScore,
            T5.Customer_BillingAddr)
   END
 FROM (   (SELECT T1.Person_Id, T1.Person_Name,  T2.Employee_Dept,
               CAST(NULL AS SqlServer.int) AS Customer_CreditScore,
               CAST(NULL AS SqlServer.nvarchar) AS Customer_BillingAddr, False AS
                _from0,
               (T2._from1 AND T2._from1 IS NOT NULL) AS _from1,  T1._from2
          FROM  ( SELECT  T.Id AS Person_Id, T.Name AS Person_Name, True AS
                  _from2
                FROM HR AS T) AS T1
                LEFT OUTER JOIN (
                  SELECT  T.Id AS Person_Id,  T.Dept AS Employee_Dept, True AS
                   _from1
                  FROM dbo.Empl AS T) AS T2
               ON T1.Person_Id = T2.Person_Id )
          UNION ALL (
               SELECT  T.Id AS Person_Id,  T.Name AS Person_Name,
                       CAST(NULL AS SqlServer.nvarchar) AS Employee_Dept,
                       T.Score AS Customer_CreditScore,  T.Addr AS
Customer_BillingAddr,
                       True AS _from0,  False AS _from1,  False AS _from2
               FROM Client AS T)
       ) AS T5
```

# Constraints → Transformations (2)

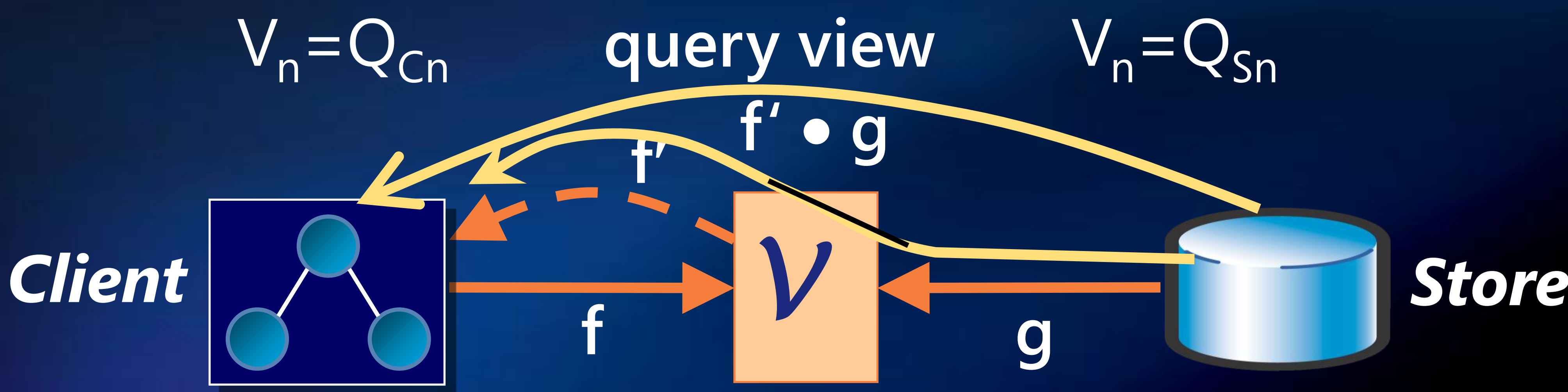
- Difficulty depends on
  - Whether the constraints are functions
  - The transformation language (e.g., SQL, XSLT)
  - Expressiveness of constraints
  - Optimization required

# Compiling Constraints

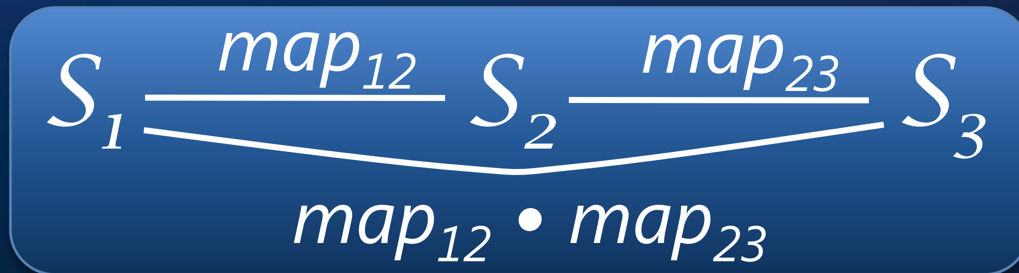
[Melnik, Adya, Bernstein SIGMOD 07]

- Mapping: $\{Q_{C1}=Q_{S1}, \ldots, Q_{Cn}=Q_{Sn}\}$
- E.g., f: SELECT p.Id, p.Name FROM **Persons** p = g: SELECT Id, Name FROM ClientInfo

- f: $V_1=Q_{C1} \cup$
  $V_2=Q_{C2} \cup$
  …
  $V_n=Q_{Cn}$
- g: $V_1=Q_{S1} \cup$
  $V_2=Q_{S2} \cup$
  …
  $V_n=Q_{Sn}$



# Composition




$\mathbb{I}(S_1)$ are the instances of schema $S_1$

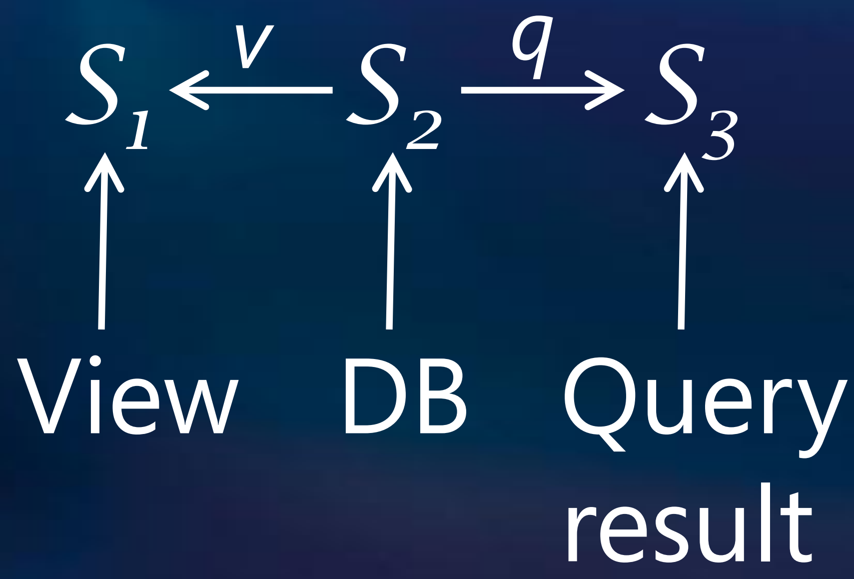
$map_{12} \subseteq \mathbb{I}(S_1) \times \mathbb{I}(S_2)$ $\quad$ $map_{13} \subseteq \mathbb{I}(S_2) \times \mathbb{I}(S_3)$

$$map_{13} = \{<d_1 \in \mathbb{I}(S_1), d_3 \in \mathbb{I}(S_3)> \mid \exists\, d_2 \in \mathbb{I}(S_2)\ (<d_1, d_2> \in map_{12}) \wedge (<d_2, d_3> \in map_{23})\ \}$$

## Well known examples

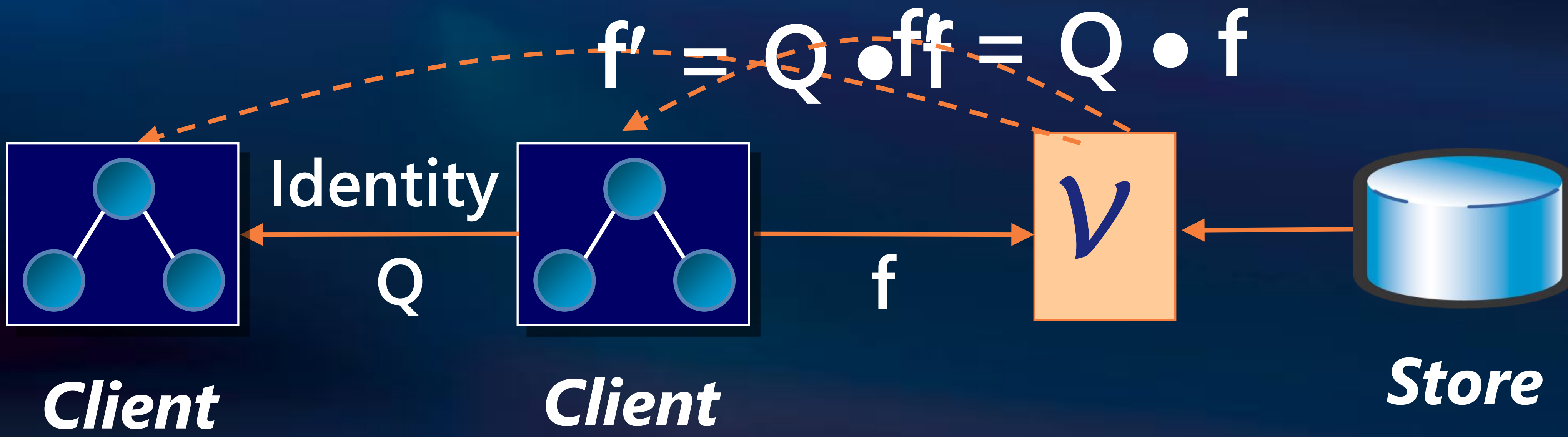
- View unfolding $S_1 \xrightarrow{v} S_2 \xrightarrow{q} S_3$
- Answering queries using views

# Answering Queries Using Views




- Goal: Rewrite q to access $S_1$ only
- There are many solutions.
  - A.Y. Halevy: Answering Queries Using Views: A Survey. *VLDB J.* 10(4): 270-294 (2001).

# Computing f' from f



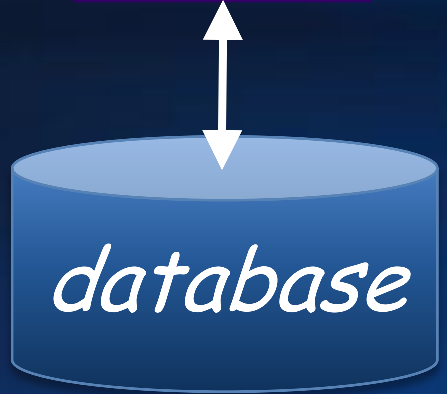

- Use query rewriting to compute f′ from f
- This is mapping composition.

# Scenarios

1. Create mappings
- ✓ Match
- ✓ ConstraintGen
- ✓ TransGen
- ✓ ModelGen

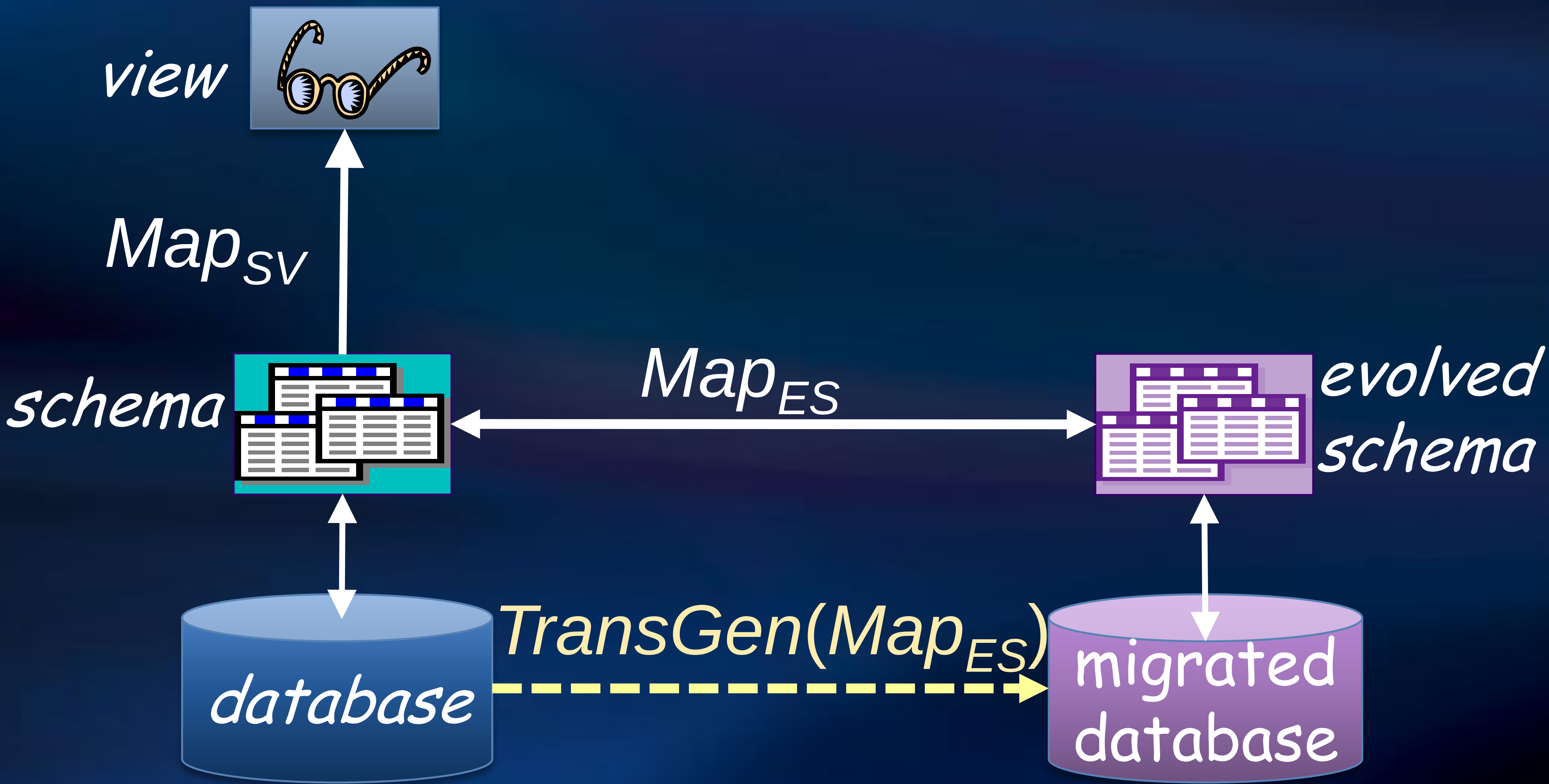
**2. Evolve mappings**
- Compose
- Diff
- Merge
- Inverse

# Schema Evolution [ Rahm, Bernstein. SIGMOD Rec. Dec 06]

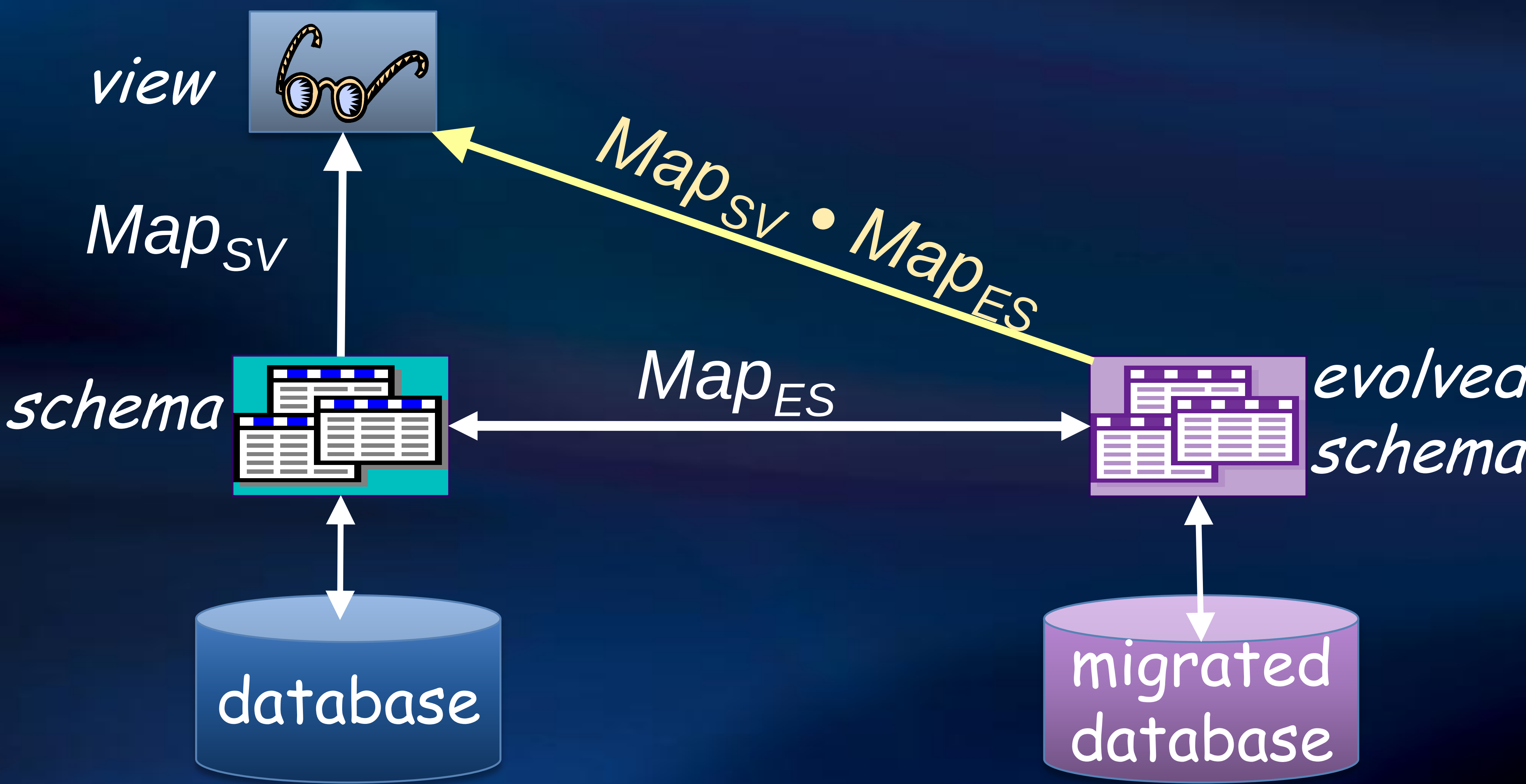

- Schema evolves
- What about database & view?

# Data Migration




1. Create mapping: *schema* ⇔ *evolved schema*
2. Generate a transformation

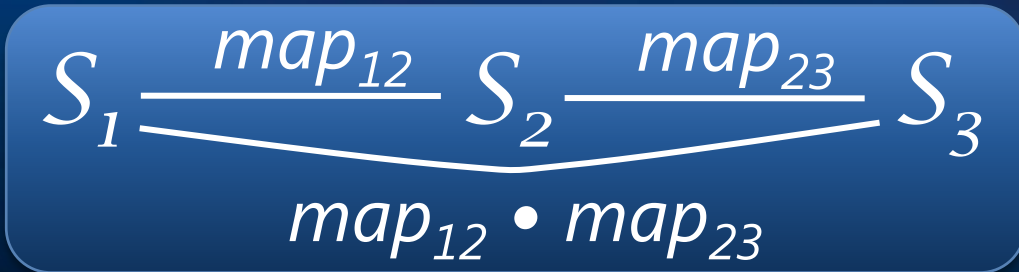
# View Migration




- Compose *$Map_{SV}$* and *$Map_{ES}$* to connect ***view*** to ***evolved schema***

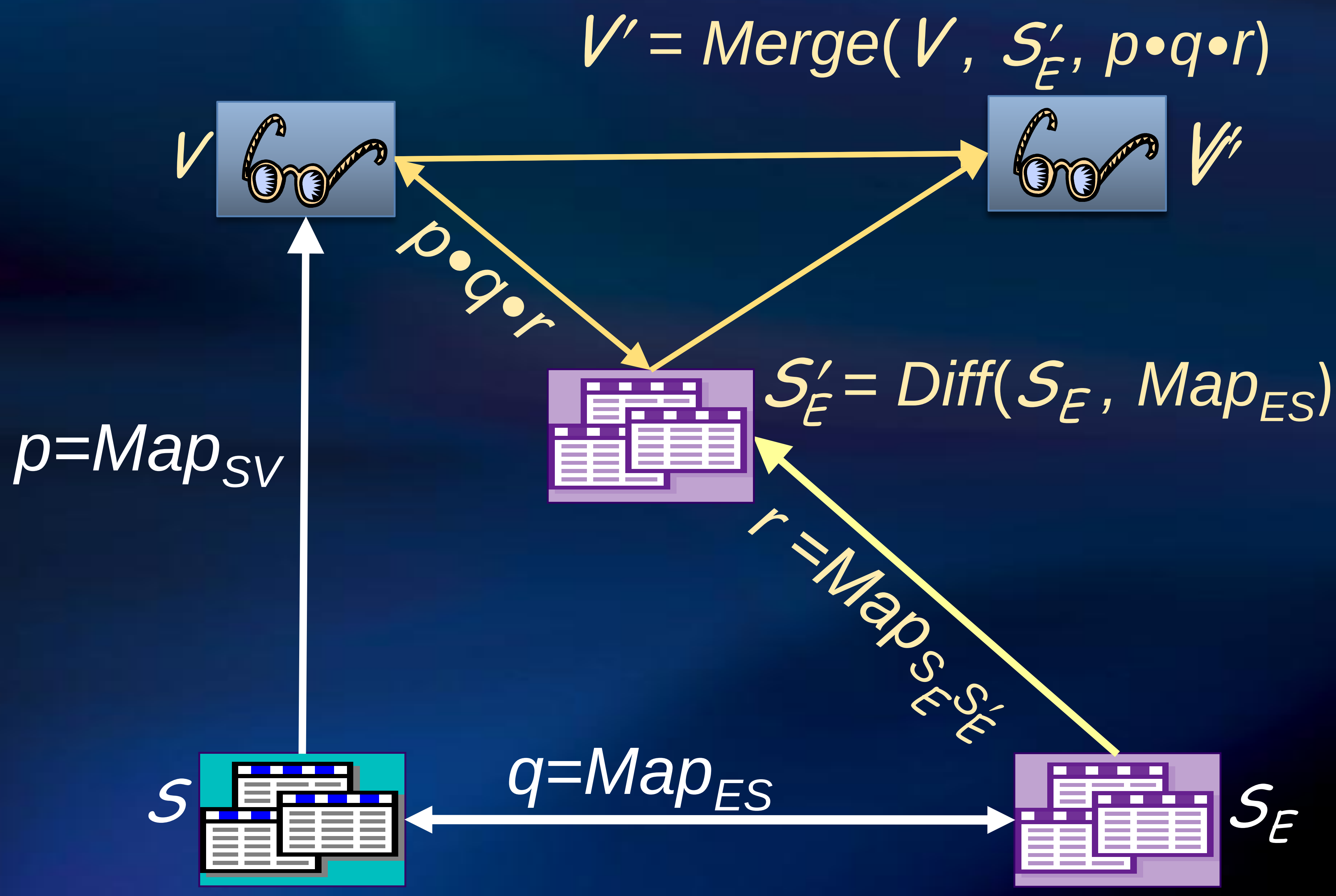
# Composition (2)

[Fagin, Kolaitis, Popa, Tan. TODS 05]
[Nash, Bernstein, Melnik. TODS 07]
[Yu, Popa. VLDB 05]
[Bernstein, Green, Melnik, Nash. VLDB 06]




- Some natural 1st-order mapping languages are not closed under composition
  - Sometimes, it's undecidable whether the composition is expressible in the input language
  - Can settle for a partial solution over 1st-order mappings
- Or you can use a 2nd-order mapping language that's closed under composition
  - There's a composition algorithm to compute it
- Some prototype implementations reported
  - Practical applications needed

# Augment View with $S_E$'s new data



# Summary

- There’s a big market looking for solutions
- There’s progress on many operators
  - But it's incomplete
  - For mappings with limited expressiveness
- Schema evolution is a particularly important scenario where the operators could have a major impact.

Microsoft®

*Your potential. Our passion.*™