

# Object-Oriented Databases

## Object Model Language: OML

- Collection Algebra
- Language Design
- Data Definition, Manipulation and Query Language



# Collection Algebra

- Operations defined for collections of the OM data model
  - union  $\cup : (\text{coll}[t_1], \text{coll}[t_2]) \rightarrow \text{coll}[t_1 \sqcup t_2]$
  - intersection  $\cap : (\text{coll}[t_1], \text{coll}[t_2]) \rightarrow \text{coll}[t_1 \cap t_2]$
  - difference  $- : (\text{coll}[t_1], \text{coll}[t_2]) \rightarrow \text{coll}[t_1]$
  - selection  $\% : (\text{coll}[t], t \rightarrow \text{boolean}) \rightarrow \text{coll}[t]$
  - map  $\propto : (\text{coll}[t_1], t_1 \rightarrow t_2) \rightarrow \text{coll}[t_2]$
  - reduce  $\oplus : (\text{coll}[t_1], ((t_1, t_2) \rightarrow t_2), t_2) \rightarrow t_2$
  - flatten  $\pm : \text{coll}[\text{coll}[t]] \rightarrow \text{coll}[t]$
- Semantics of operations depend on collection behaviour
  - set theory defines semantics for set collections
  - generalisation for bag, sequences and rankings
  - mixing and converting of collection types

# Collection Algebra for Bags

- Bags can be represented as sets of tuples
  - $B = \langle x, y, y, z, z, z \rangle$
  - $B = \{ (x, 1), (y, 2), (z, 3) \}$
- Membership
  - $x$  is a member of bag  $B$  if  $x$  occurs one or more times
  - $x \in_{\text{bag}} B \iff \exists n \in \mathbb{N}^+ : (x, n) \in_{\text{set}} B$

# Bag Union

## ■ Definition

- $B_1 \cup_{\text{bag}} B_2 = \{ (x, n) \mid \exists n_1, n_2 : ((x, n_1) \in_{\text{set}} B_1 \wedge (x, n_2) \in_{\text{set}} B_2 \wedge n = \max(n_1, n_2)) \}$

## ■ Example

- $B_1 = \langle 2, 2, 2, 4, 4, 5 \rangle$
- $B_2 = \langle 1, 2, 4, 4 \rangle$
- $B_1 \cup_{\text{bag}} B_2 = \langle 1, 2, 2, 2, 4, 4, 5 \rangle$

# Bag Addition

## ■ Definition

- $B_1 \uplus B_2 = \{ (x, n) \mid \exists n_1, n_2 : ((x, n_1) \in_{\text{set}} B_1 \wedge (x, n_2) \in_{\text{set}} B_2 \wedge n = n_1 + n_2) \}$

## ■ Example

- $B_1 = \langle 2, 2, 2, 4, 4, 5 \rangle$
- $B_2 = \langle 1, 2, 4, 4 \rangle$
- $B_1 \uplus B_2 = \langle 1, 2, 2, 2, 2, 4, 4, 4, 4, 5 \rangle$

# Bag Intersection

## ■ Definition

- $B_1 \cap_{\text{bag}} B_2 = \{ (x, n) \mid \exists n_1, n_2 : ((x, n_1) \in_{\text{set}} B_1 \wedge (x, n_2) \in_{\text{set}} B_2 \wedge n = \min(n_1, n_2)) \}$

## ■ Example

- $B_1 = \langle 2, 2, 2, 4, 4, 5 \rangle$
- $B_2 = \langle 1, 2, 4, 4 \rangle$
- $B_1 \cap_{\text{bag}} B_2 = \langle 2, 4, 4 \rangle$

# Bag Difference

## ■ Definition

- $B_1 -_{\text{bag}} B_2 = \{ (x, n) \mid \exists n_1 : ((x, n_1) \in_{\text{set}} B_1 \wedge ((x \notin_{\text{bag}} \wedge n = n_1) \vee \exists n_2 : ((x, n_2) \in_{\text{set}} B_2 \wedge n_1 > n_2 \wedge n = n_1 - n_2))) \}$

## ■ Example

- $B_1 = \langle 2, 2, 2, 4, 4, 5 \rangle$
- $B_2 = \langle 1, 2, 4, 4 \rangle$
- $B_1 -_{\text{bag}} B_2 = \langle 2, 2, 5 \rangle$

# Bag Selection

## ■ Definition

- $B \%_{\text{bag}} P = \{ (x, n) \mid (x, n) \in_{\text{set}} B \wedge P(x) = \text{true} \}$

## ■ Example

- $B = \langle 2, 2, 4, 4, 4, 5 \rangle$
- $P = x \rightarrow (x \bmod 2 = 0)$
- $B \% P = \langle 2, 2, 4, 4, 4 \rangle$

# Bag Map

## ■ Definition

$$\blacksquare B \propto_{\text{bag}} f = \bigoplus_{\text{set}} B((x, n), B_1) \rightarrow (\{ (f(x), n) \} \uplus B_1) \emptyset_{\text{bag}}$$

## ■ Example

$$\blacksquare B = \langle -2, -2, 1, 2 \rangle = \{ (-2, 2), (1, 1), (2, 1) \}$$

$$\blacksquare f = x \rightarrow x^2$$

$$\blacksquare B \propto_{\text{bag}} f = \langle 4, 4, 4, 1 \rangle$$

## ■ Execution of map operation

$$\emptyset_{\text{bag}} \uplus \langle 4, 4 \rangle \uplus \langle 1 \rangle \uplus \langle 4 \rangle = \langle 4, 4, 4, 1 \rangle$$

# Bag Reduce

## ■ Definition

- $\bigoplus_{\text{bag}} B \text{ f } v =$  if  $B = \emptyset_{\text{bag}}$  then  $v$   
else  $f(x, \bigoplus_{\text{bag}} B' \text{ f } v)$  where  $B = B' \uplus \{ (x, 1) \}$

## ■ Example

- $B = \langle 2, 2, 4, 4, 4, 5 \rangle$
- $f = (x, y) \rightarrow x + y$
- $v = 0$
- $\bigoplus_{\text{bag}} B \text{ f } v = 21$

# Bag Flatten

## ■ Definition

- $\pm B = \bigoplus_{\text{bag}} B(x, y) \rightarrow (\text{ext}(x) \uplus y) \emptyset_{\text{bag}}$

## ■ Example

- $B = \langle \langle 1, 1, 2 \rangle, \langle 1, 1, 2 \rangle, \langle 2, 2, 3 \rangle \rangle$

- $\pm B = \langle 1, 1, 1, 1, 2, 2, 2, 2, 3 \rangle$

# Collection Algebra for Ordered Collections

- As an exercise, consider how the model could be extended in terms of operations for ordered collections
- Sequences
  - $L_1 \cup_{\text{seq}} L_2: [7, 3, 1, 3] \cup_{\text{seq}} [3, 2, 2] = ?$
  - $L_1 \cap_{\text{seq}} L_2: [7, 3, 1, 3] \cap_{\text{seq}} [3, 2, 2] = ?$
  - $L_1 -_{\text{seq}} L_2: [7, 3, 1, 3] -_{\text{seq}} [3, 2, 2] = ?$
- Rankings
  - $R_1 \cup_{\text{rnk}} R_2: |7, 3, 1, 4| \cup_{\text{rnk}} |2, 4, 3| = ?$
  - $R_1 \cap_{\text{rnk}} R_2: |7, 3, 1, 4| \cap_{\text{rnk}} |2, 4, 3| = ?$
  - $R_1 -_{\text{rnk}} R_2: |7, 3, 1, 4| -_{\text{rnk}} |2, 4, 3| = ?$

# Object Model Language (OML)

- Declarative object-oriented language for OM data model
- OML Data Definition Language
  - object and structured type definition
  - method definition and implementation
  - collection, association and constraint definition
- OML Data Manipulation Language
  - creating and updating of objects
  - create, update, delete, dress and strip operations
- OML Query Language
  - expressions and functions for values of base types
  - operations on objects to access attributes and invoke methods
  - operations on collections based on collection algebra

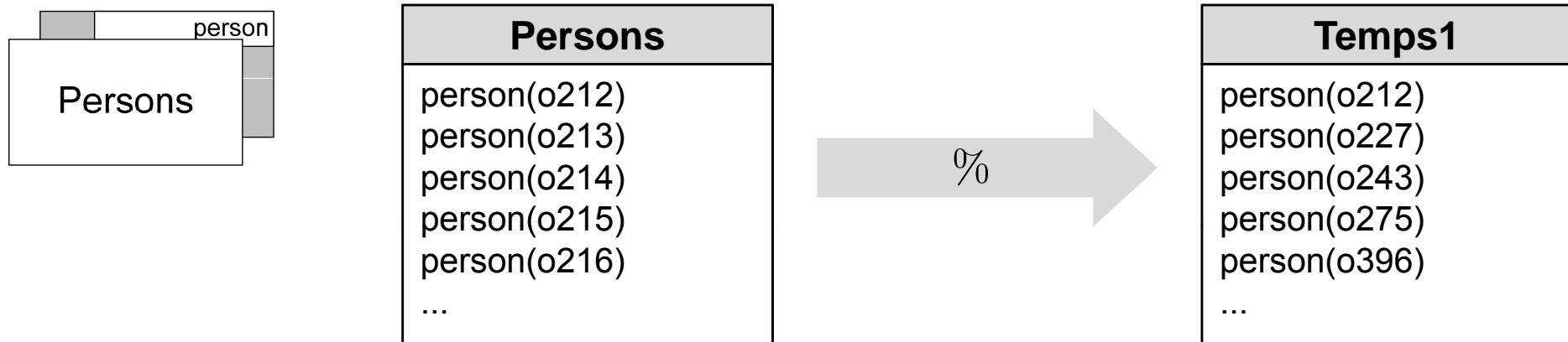
# Type Schema Definition

```
create type contact (  
  name: string,  
  phone: string,  
  fax: string,  
  email: uri,  
  www: uri  
);  
create type person subtype of contact (  
  title: string,  
  photo: uri,  
  method getWorkPlaces() returns (locations: set of location) (  
    return ran((WorksFor compose SituatedAt) dr  
      (all $p in Persons having ($p = this))  
    )  
  )  
);  
create type organisation subtype of contact (  
  description: string  
);
```

# Classification Schema Definition

```
// Collections
create collection Contacts as set of contact;
create collection Persons as set of person;
create collection Organisations as set of organisation;
// Binary Collections
create collection WorksFor as set of (person, organisation);
// Constraints
create constraint assoc_WorksFor
    association on WorksFor from Persons (0,*) to Organisations (0,*);
create constraint subc_Persons
    subcollection Persons restricts Contacts;
create constraint subc_Organisations
    subcollection Organisations restricts Contacts;
create constraint part_Contacts
    classification (Persons, Organisations) partition Contacts;
create constraint k_Persons classification Persons is kind;
create constraint k_Organisations classification Organisation is kind;
```

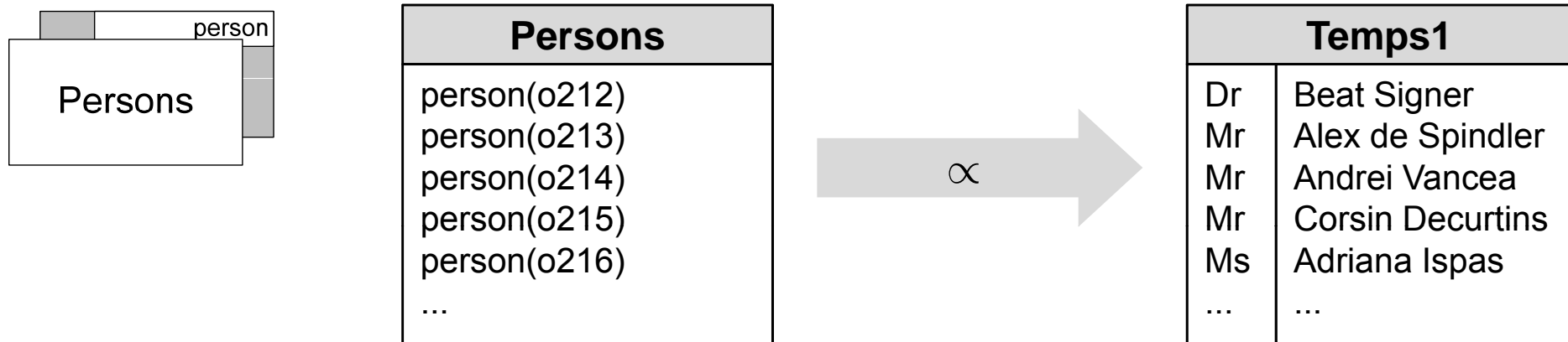
# Selection



- Select all objects in a collection matching a predicate
- $\% : (\text{coll}[t], t \rightarrow \text{boolean}) \rightarrow \text{coll}[t]$

```
all $p in Persons having ($p.title = "Prof")
```

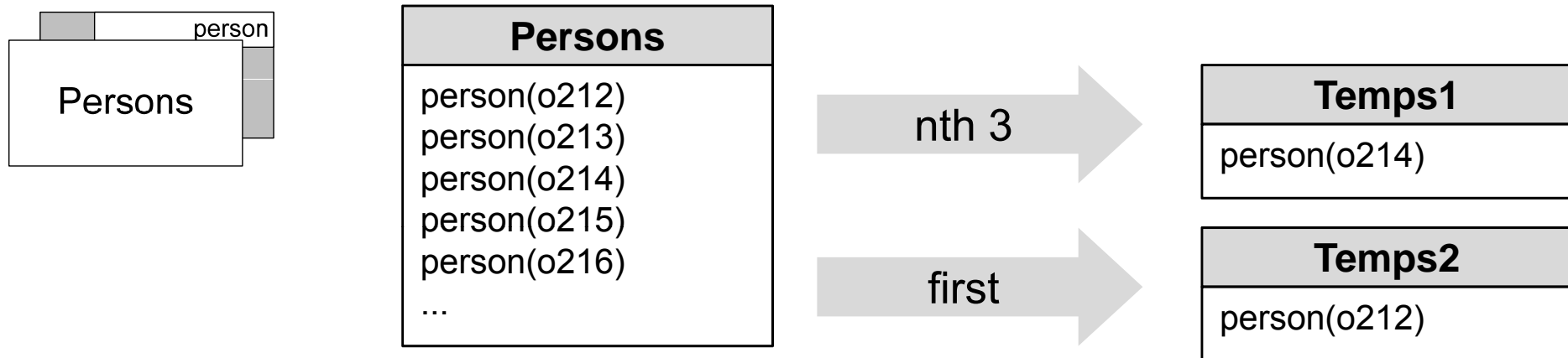
# Map



- Apply an operation to every object in a collection
- $\infty : (\text{coll}[t_1], t_1 \rightarrow t_2) \rightarrow \text{coll}[t_2]$

```
map $p in Persons by ($p.title x $p.name)
```

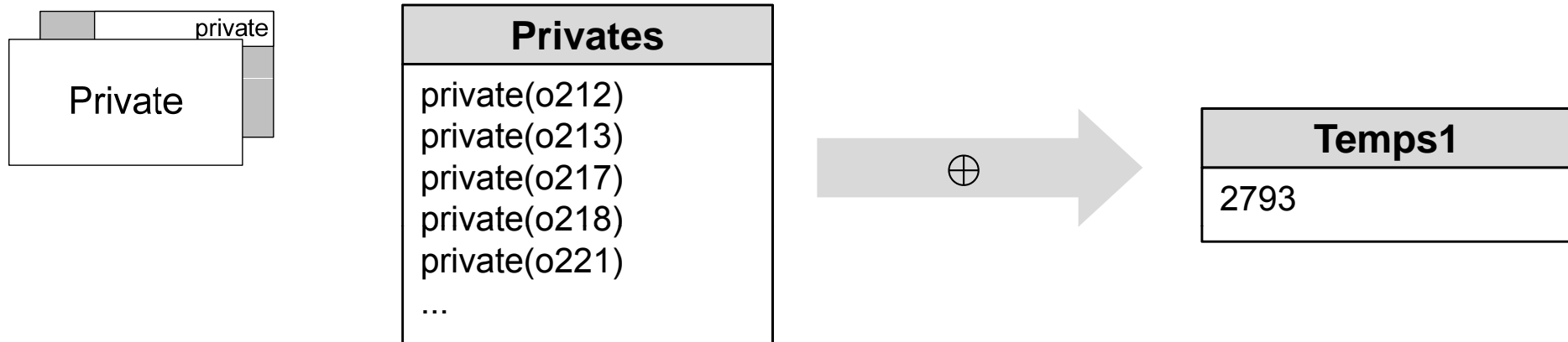
# Element Extraction



- Extract an element from a collection
- $\text{nth} : (\text{coll}[t], \mathbb{Z}) \rightarrow t$

```
the 3 in Persons  
first Persons; last Persons  
max Privates.getAge().years  
min Privates.getAge().years
```

# Reduce



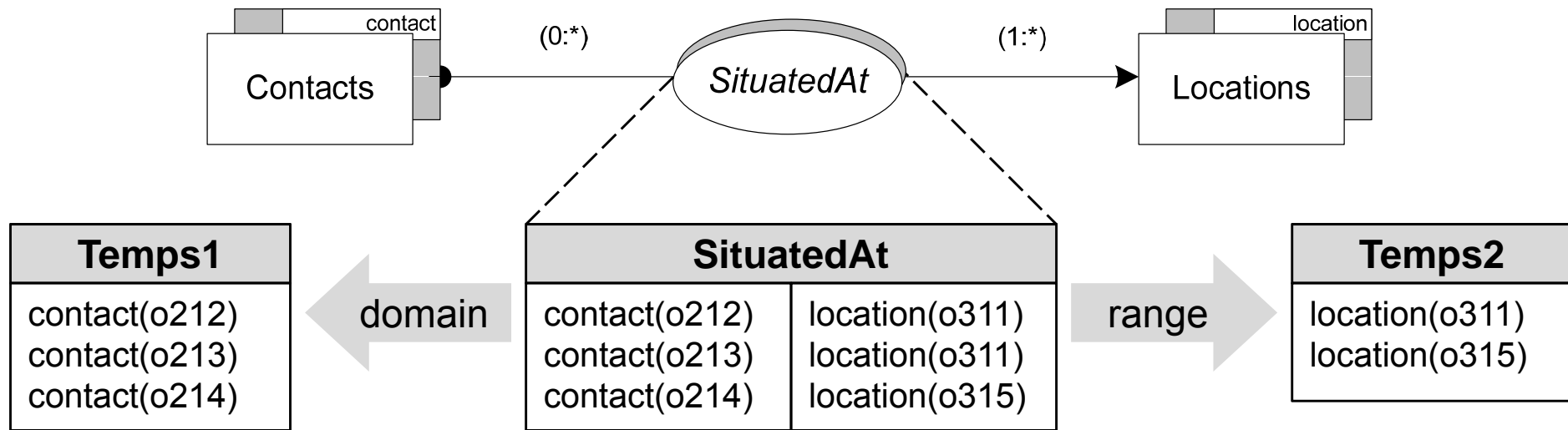
- Calculate an aggregate value
- $\oplus : (\text{coll}[t_1], ((t_1, t_2) \rightarrow t_2), t_2) \rightarrow t_2$

```
reduce $p in Privates
  aggregate $a by (($p.getAge()).years + $a)
  default 0
```

# Binary Collections

- All operations applicable to collections can be applied to binary collection
- Binary collections support additional operations
  - domain and range
  - domain and range restriction as well as subtraction
  - inverse
  - nest
  - compose
  - closure
  - division

# Domain and Range

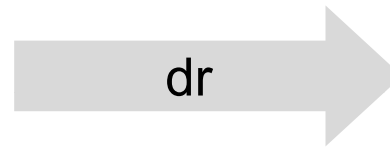


- $\text{dom} : \text{coll}[(t_1, t_2)] \rightarrow \text{coll}[t_1]$   
 $\text{ran} : \text{coll}[(t_1, t_2)] \rightarrow \text{coll}[t_2]$

`domain SitedAt`  
`range SitedAt`

# Domain Restriction

SituatedAt	
contact(o212)	location(o311)
contact(o213)	location(o311)
contact(o214)	location(o315)
contact(o215)	location(o321)
contact(o216)	location(o347)



Temps1	
contact(o215)	location(o321)

- Restrict the domain of a binary collection
- $dr : (coll[(t_1, t_2)], coll[t_3]) \rightarrow coll[(t_1, t_2)]$

```
SituatedAt
  dr (
    all $c in Contacts having ($c.name = "ETH Zurich")
  )
```

# Domain Subtraction

SituatedAt	
contact(o212)	location(o311)
contact(o213)	location(o311)
contact(o214)	location(o315)
contact(o215)	location(o321)
contact(o216)	location(o347)



Temps1	
contact(o212)	location(o311)
contact(o213)	location(o311)
contact(o214)	location(o315)
contact(o216)	location(o347)

- Restrict the domain of a binary collection
- $ds : (coll[(t_1, t_2)], coll[t_3]) \rightarrow coll[(t_1, t_2)]$

```
SituatedAt
  ds (
    all $c in Contacts having ($c.name = "ETH Zurich")
  )
```

# Range Restriction

SituatedAt	
contact(o212)	location(o311)
contact(o213)	location(o311)
contact(o214)	location(o315)
contact(o215)	location(o321)
contact(o216)	location(o347)



Temps1	
contact(o214)	location(o315)
contact(o215)	location(o321)

- Restrict the range of a binary collection
- $rr : (\text{coll}[(t_1, t_2)], \text{coll}[t_3]) \rightarrow \text{coll}[(t_1, t_2)]$

```
SituatedAt
  rr (
    all $l in Locations having ($l.city = "Zurich")
  )
```

# Range Subtraction

SituatedAt	
contact(o212)	location(o311)
contact(o213)	location(o311)
contact(o214)	location(o315)
contact(o215)	location(o321)
contact(o216)	location(o347)

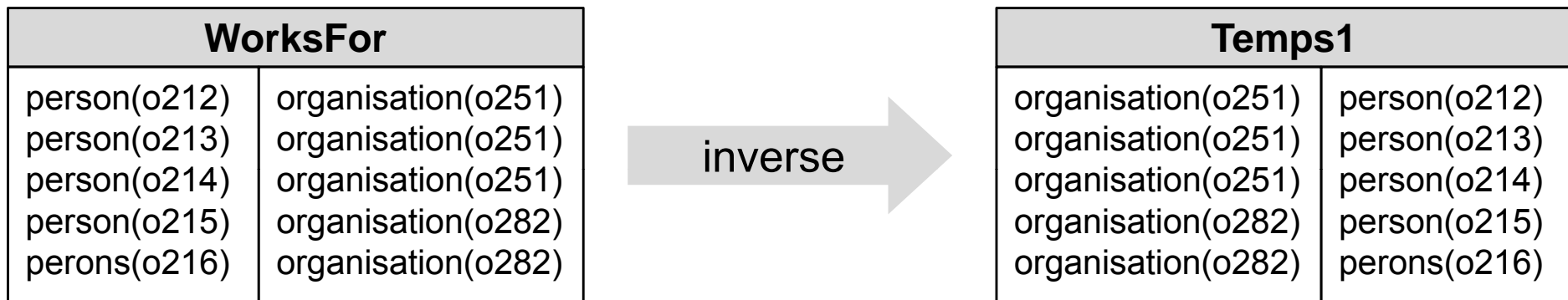


Temps1	
contact(o212)	location(o311)
contact(o213)	location(o311)
contact(o216)	location(o347)

- Restrict the range of a binary collection
- $rs : (coll[(t_1, t_2)], coll[t_3]) \rightarrow coll[(t_1, t_2)]$

```
SituatedAt
  rs (
    all $1 in Locations having ($1.city = "Zurich")
  )
```

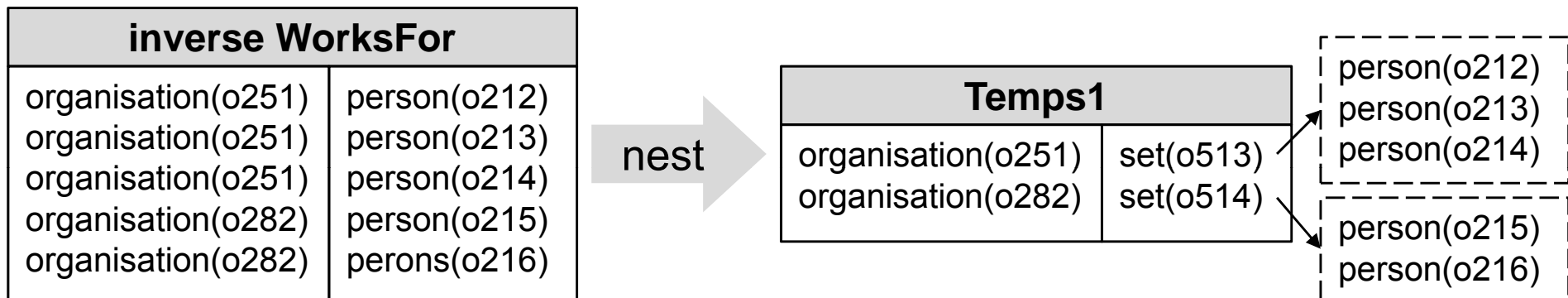
# Inverse



- Swapping domain and range of a binary collection
- $\text{inv} : \text{coll}[(t_1, t_2)] \rightarrow \text{coll}[(t_2, t_1)]$

`inverse WorksFor`

# Nest

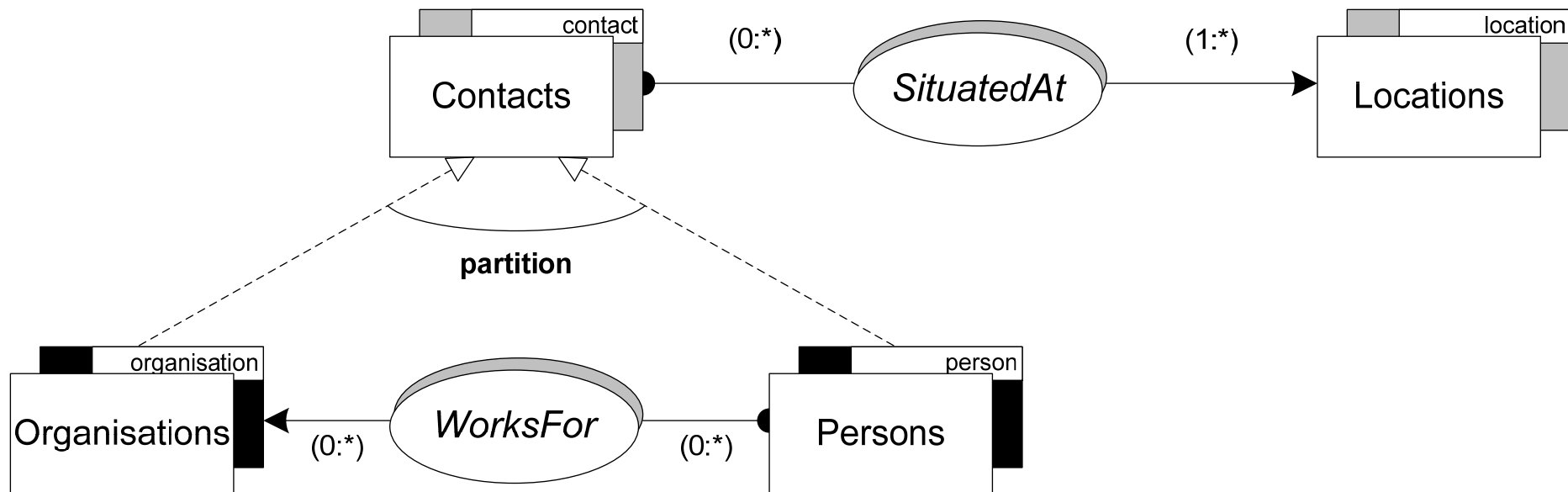


- Grouping of binary collections
- $\text{nest} : \text{coll}[(t_1, t_2)] \rightarrow \text{coll}[(t_1, \text{coll}[t_2])]$

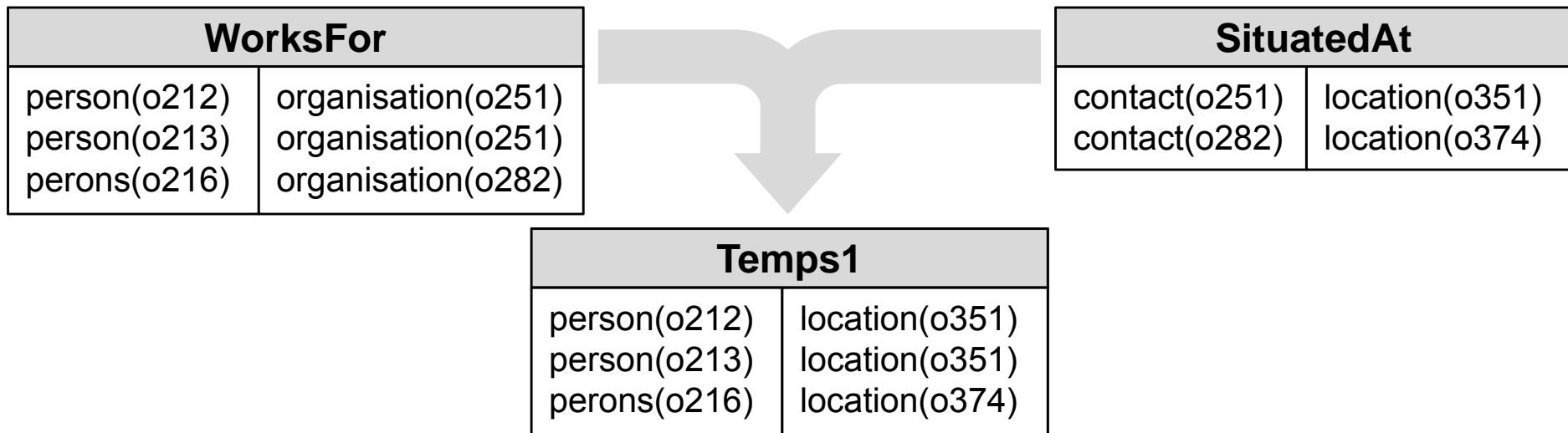
```
nest (inverse WorksFor)
```

# Compose

- To find associated objects in the database, it may be necessary to compose (join) associations
- Find the possible work locations of persons?



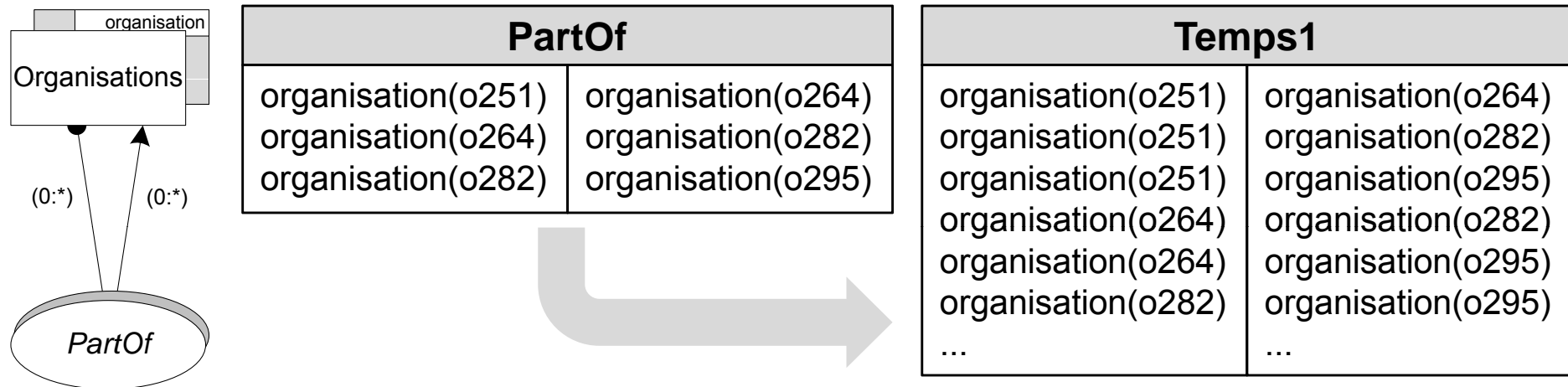
# Compose



- Swapping domain and range of a binary collection
- $\circ : (\text{coll}[(t_1, t_2)], \text{coll}[(t_3, t_4)]) \rightarrow \text{coll}[(t_1, t_4)]$

WorksFor **compose** SituatingAt

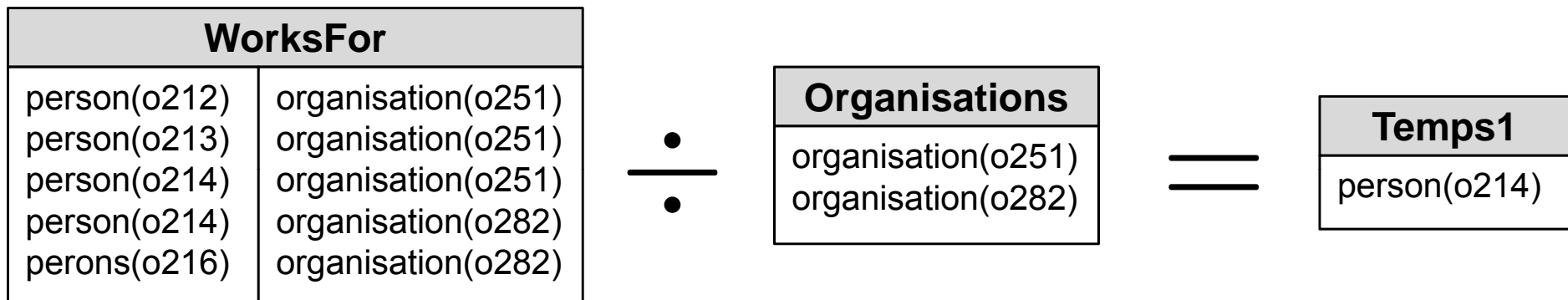
# Closure



- Repeatedly compose binary collection with itself
- $*$  :  $\text{coll}[(t_1, t_2)] \rightarrow \text{coll}[(t_1, t_2)]$

`closure PartOf`

# Division



- Divide a binary collection with a unary collection
- $\text{div} : (\text{coll}[(t_1, t_2)], \text{coll}[t_3]) \rightarrow \text{coll}[t_1]$

WorksFor **div** Organisations

## Literature

- Alain P. Würigler: **OMS Development Framework: Rapid Prototyping for Object-Oriented Databases**, *PhD Thesis, ETH No. 13512, 2000*

## Next Week

### Design and Implementation: OMS Avon

- Architecture: Storage, Model and Interface Layer
- Object Identifiers
- Query Processor

