ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Object-Oriented Databases

Storage and Indexing

- Type Hierarchy Indexing
- Aggregation Path Indexing
- Collection Operations

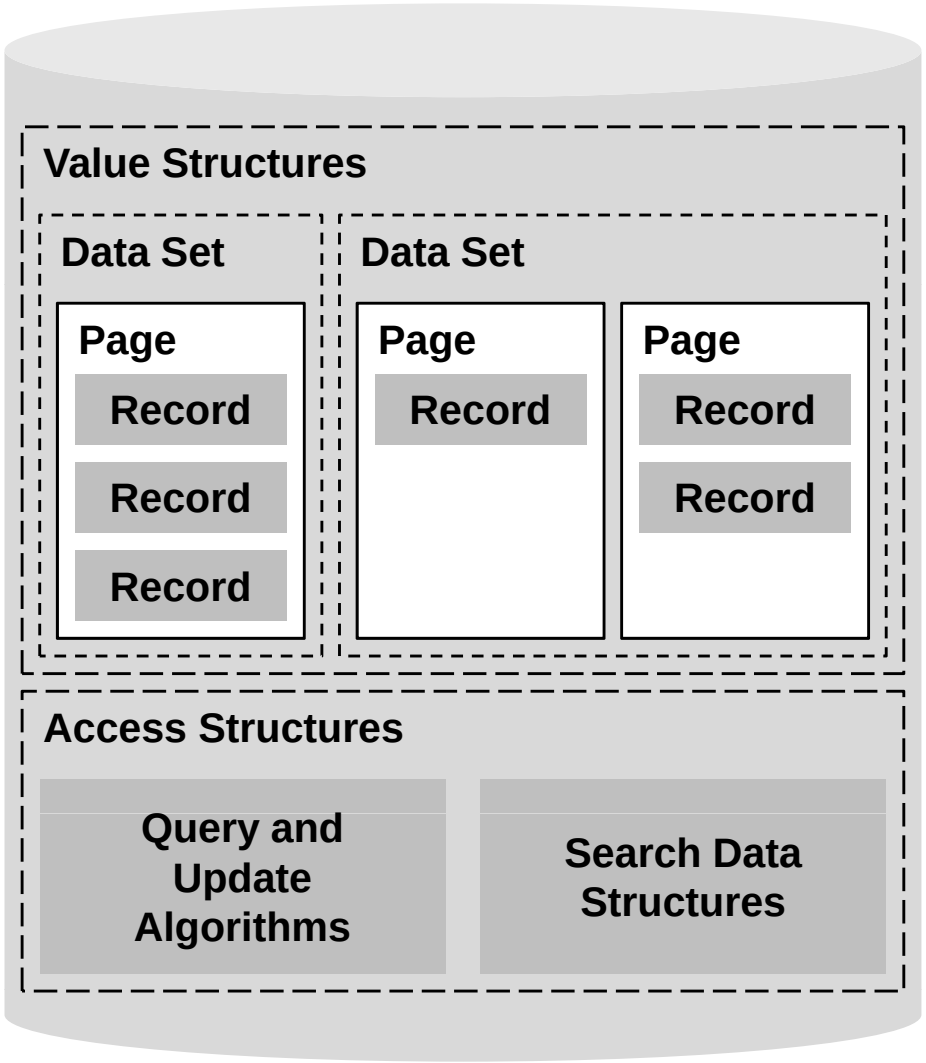
Motivation

- Management of large data sets on persistent storage
  - physical storage management
  - content-based access
- Requirements of object-oriented model surpasses those of relational model
  - storage, clustering and management of complex objects on physical persistent media
  - access through object references and query predicates
  - type inheritance hierarchies
  - relationships
  - multi-valued properties and collections
- Additional storage and indexing technologies necessary

Object-Oriented Storage

- Object-oriented storage layouts are often not very different from relational systems
- Difference stems from the algorithms used to managed physical storage
  - data structures to represent complex objects
  - grouping or clustering records of complex objects
  - grouping or clustering of referenced objects
  - management of free space
  - management of buffers

Storage Model



- Physical storage is partitioned in value and access structures
- Data managed in data sets
  - records of identical type
  - attribute set $A = \{A_1,...,A_n\}$
  - domains $D_i = \{min_i, max_i\}$
- Pages or clusters correspond to a disk block or a sequence of disk blocks of fixed size *b*
- Records are stored in pages
  - list of *n* values $t = (v_1,...,v_n)$
- Addresses reference pages
- Functions to map records to pages and vice-versa

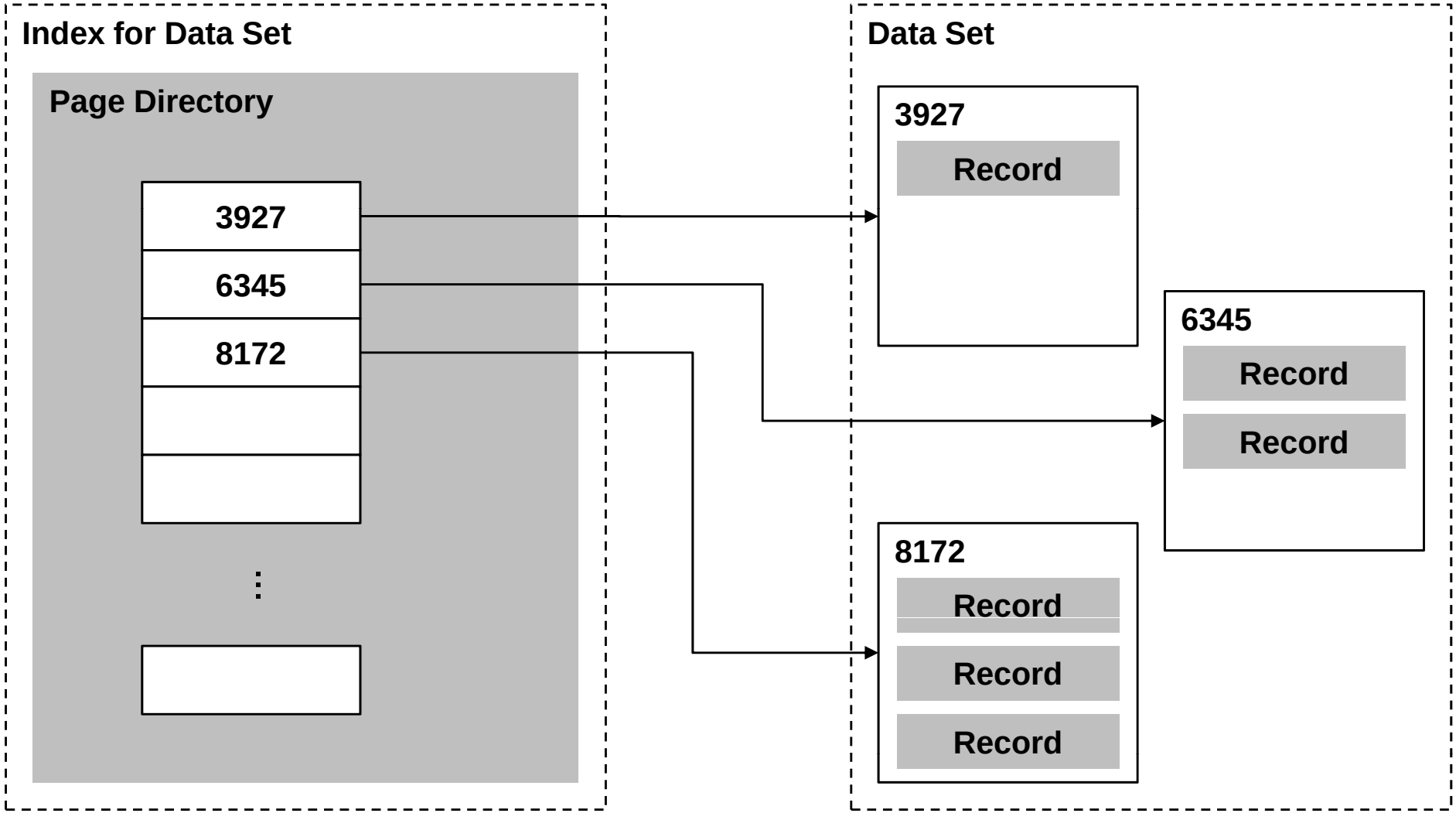
Terminology

- Query
  - **point and range queries**: match over exact values or intervals
  - **one- and multi-dimensional**: one or more matching conditions
- Index
  - **unique and non-unique**: index over keys or non-key fields (primary or secondary index)
  - **sequential and non-sequential key**: index over ordered or unordered values
  - **one- and multi-dimensional**: index over one or more fields
  - **compound**: one-dimensional index over more than one value by concatenating fields
  - **placing (clustering) and non-placing (non-clustering)**: search data structure that does (or does not) allocate record physically

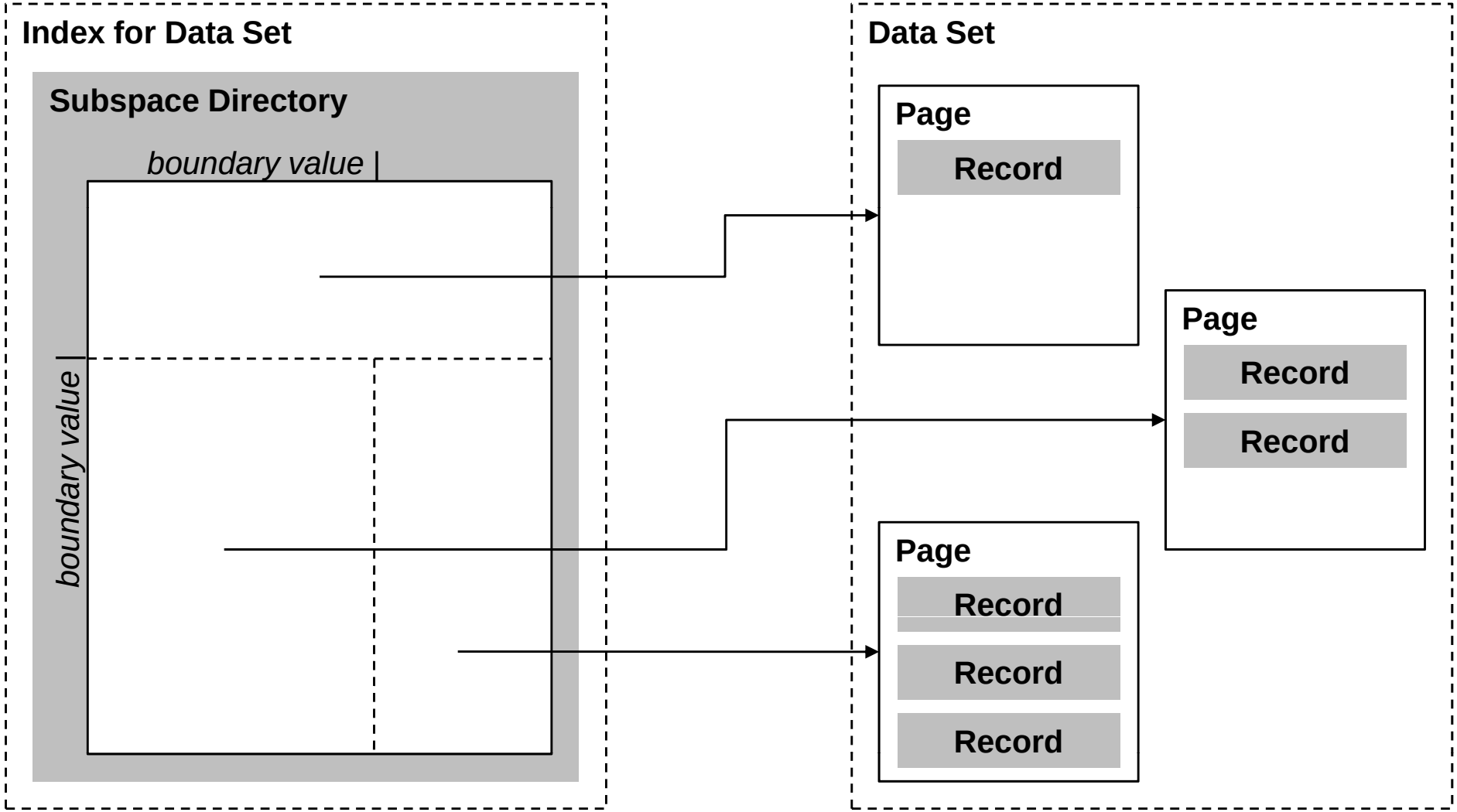
Index Data Structure Overview

- Sequential organisation
  - maintains list of storage pages that belong to data set
  - new records placed in most recently allocated storage chunk
  - during query all pages have to be retrieved
- Subspace mapping
  - decomposition of data space into subspace
  - overlapping and non-overlapping subspaces
  - B-trees, K-d trees, grid files
- Point mapping
  - direct mapping of data space elements to storage chunks
  - function determines record signature used to find address
  - hashing, extensible hashing

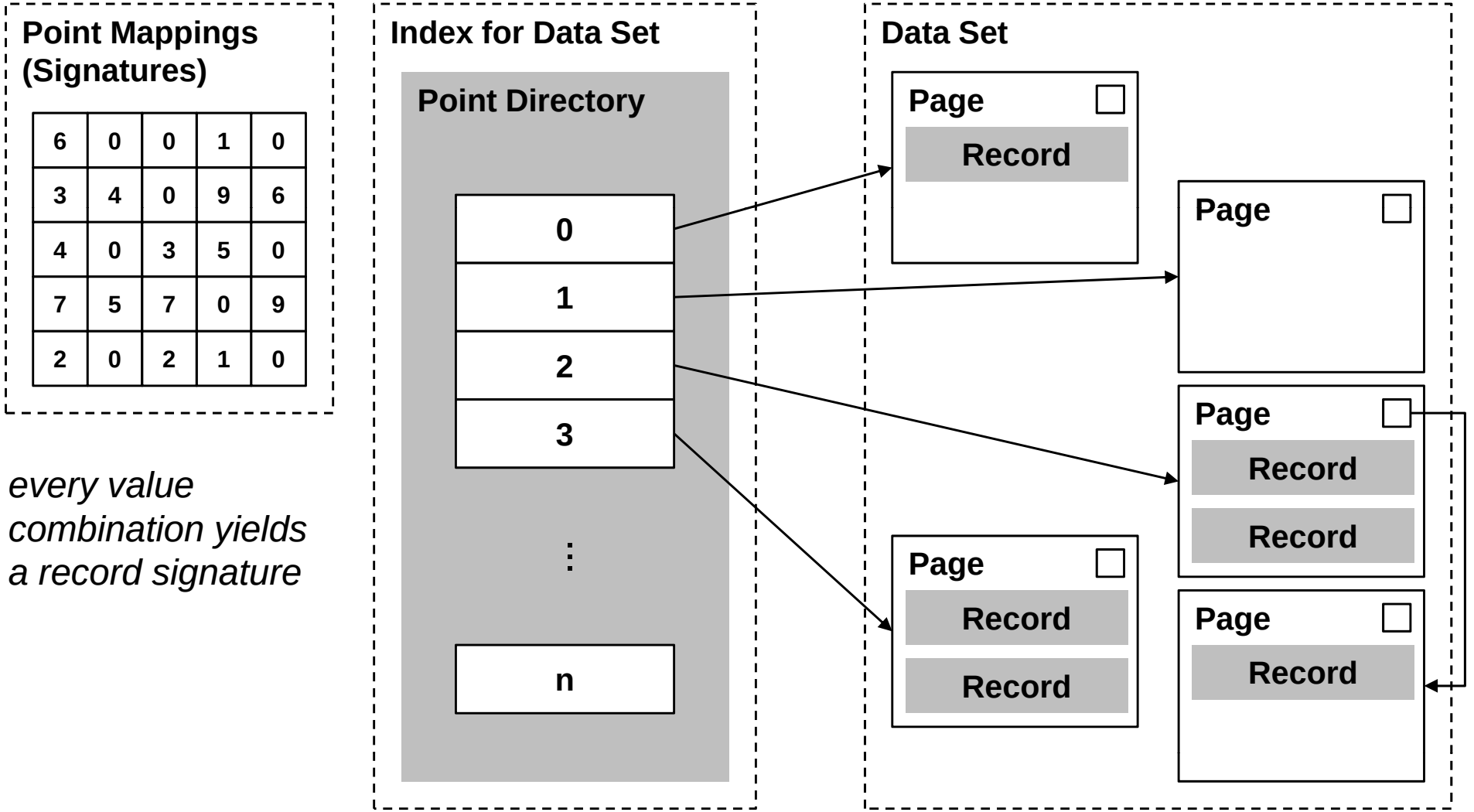
Sequential Mapping



Subspace Mapping



Point Mapping

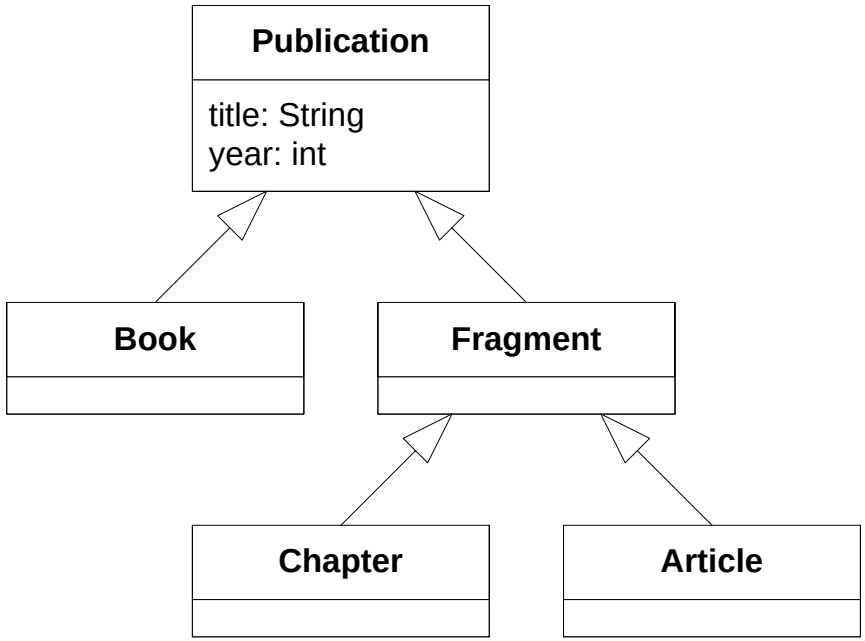


Existing Index Data Structures

- One-dimensional index data structures
  - B-tree, B^+-tree
  - extensible and linear hashing
  - bounded disorder files
  - signature files
  - partial indices
- Multi-dimensional index data structures
  - K-d tree, R-tree, H-tree, hB-tree, Quadtree, TV-tree, cell tree
  - grid files
- Architecture and Implementation of Database Systems
  - lecture covers many of these data structures
  - http://www.dbis.ethz.ch/education/ws0607/dbs_impl

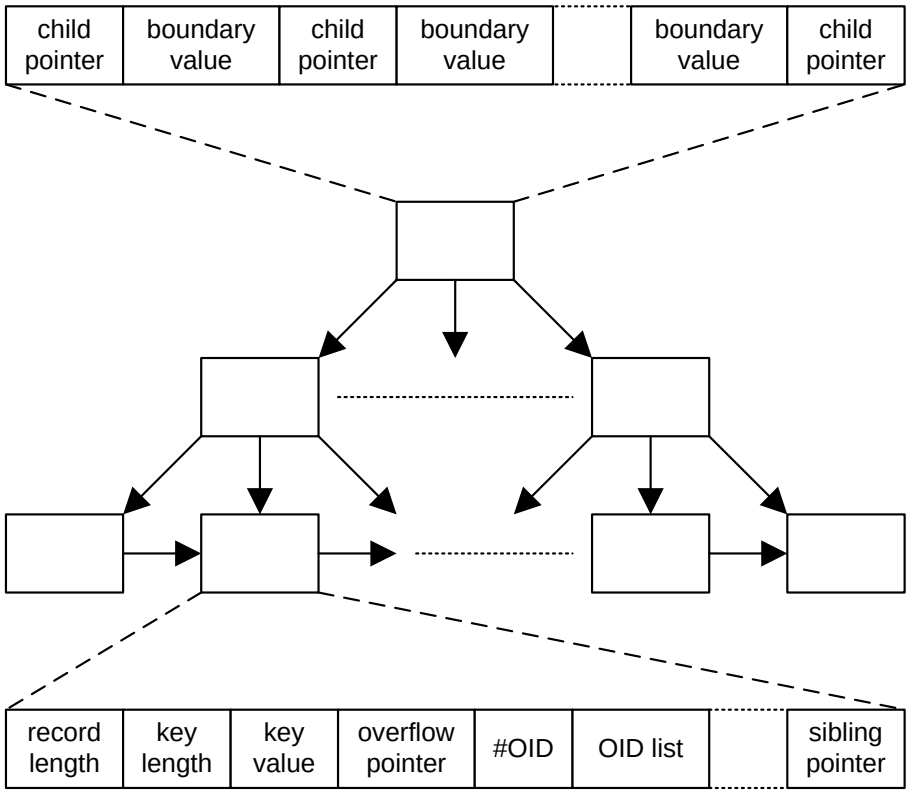
Type Hierarchy Indexing

- Object-oriented queries
  - directly reference one type
  - implicitly refer to a sub-hierarchy, i.e. a set of types
- Two design approaches for type hierarchy index data structures
  - type grouping: first-level order criterion is object type
  - key grouping: top-level data structure organises key values
- Index design has influence on resulting I/O performance of point and range queries



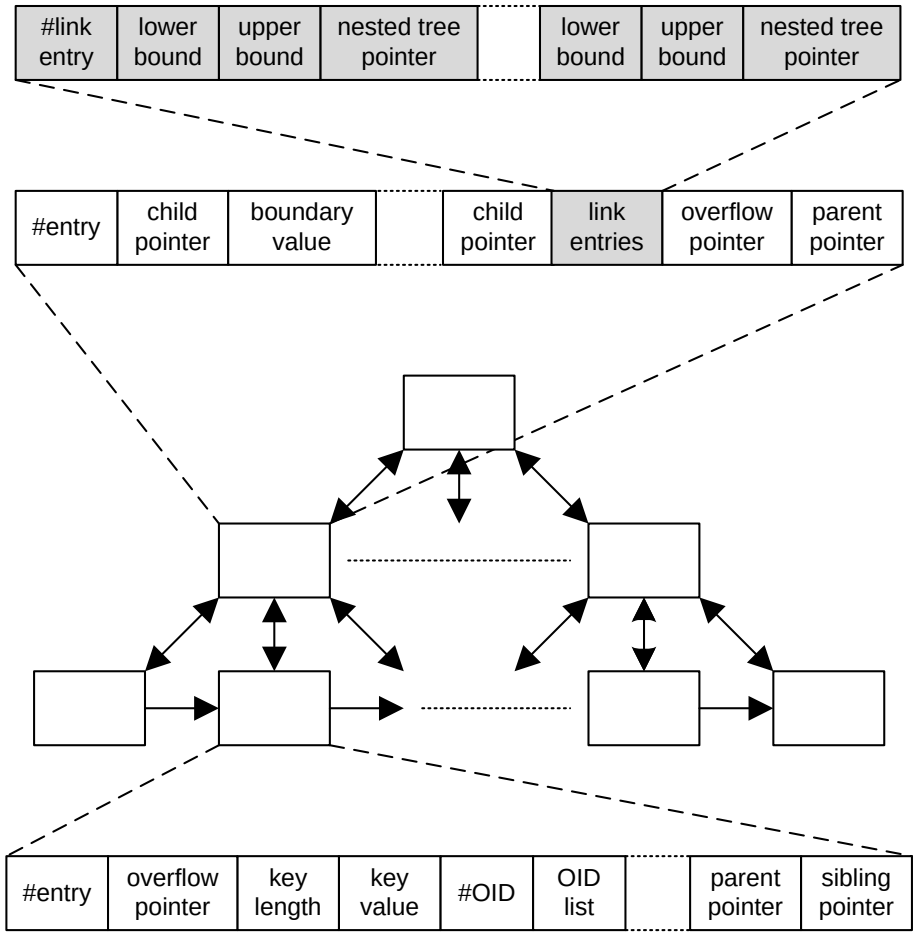
Single Class Index (SC-Index)

- Originally introduced by the ORION system in 1989
- Index construction for an attribute of a type *t*
  - construct a search structure for all types in sub-hierarchy of *t*
  - search data structures called SC-Index components
  - evaluator needs to traverse all components referenced by query
- Usually implemented using B$^+$-trees, other data structures could be used



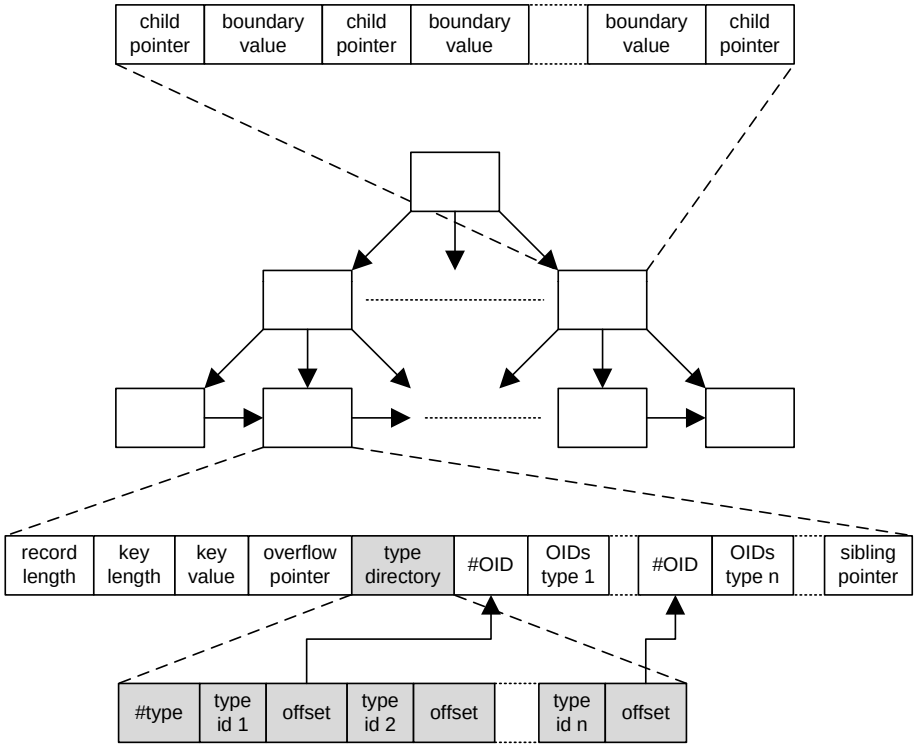
H-Tree

- A H-tree consists of a set of nested B$^+$-trees
- Nesting reflects structure of indexed type hierarchy
  - each H-tree component of indexed type is nested with H-trees of immediate subtypes of indexed type
  - H-tree index for an attribute of inheritance sub-graph is H-tree hierarchy nested according to supertype-subtype relation
- Aims to avoid full scans of each H-tree component when several types are queried



Class Hierarchy Index (CH-Index)

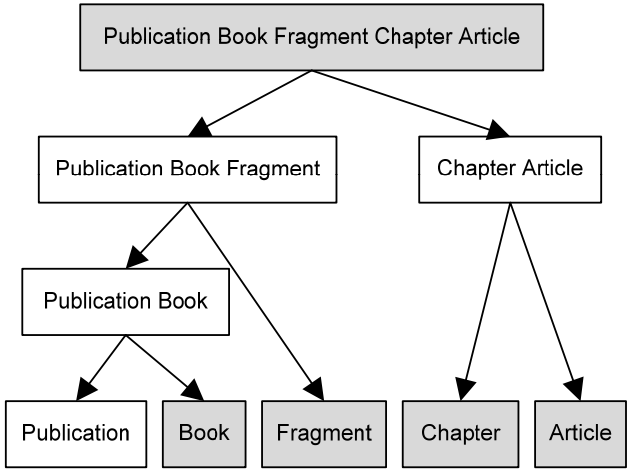
- Maintains only one search structure for all objects of all types of indexed hierarchy
- Evaluator scans through B^+-tree once (single-scan)
  - selects OIDs of types referenced by query
  - discards other OIDs
- Point queries always perform good, range queries depend on number of referenced types
  - good when queries aim at indexed type and all subtypes
  - bad if only few types of indexed hierarchy hit by the query



Class Division (CD-Index)

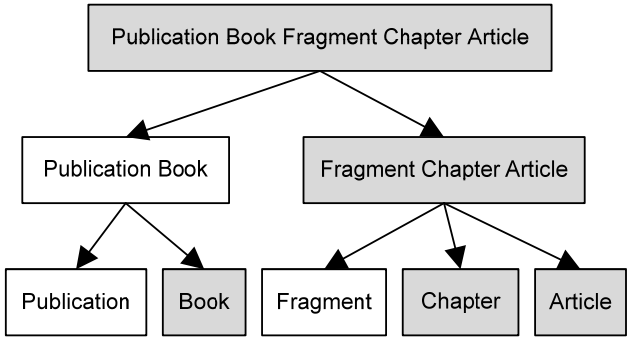
- Find a compromise between
  - indexing and storing instance set for each type
  - indexing and storing extent for each type
- Maintains a specific family of type sets for indexed hierarchy
- Each type set is managed using a search data structure
- Parameters q and r give upper bounds for decompositions
  - q: number of search structures required to build a type extent
  - r: number of times a type set is managed redundantly

- CD-index with space pruning



$q = 3$
$r = 2$

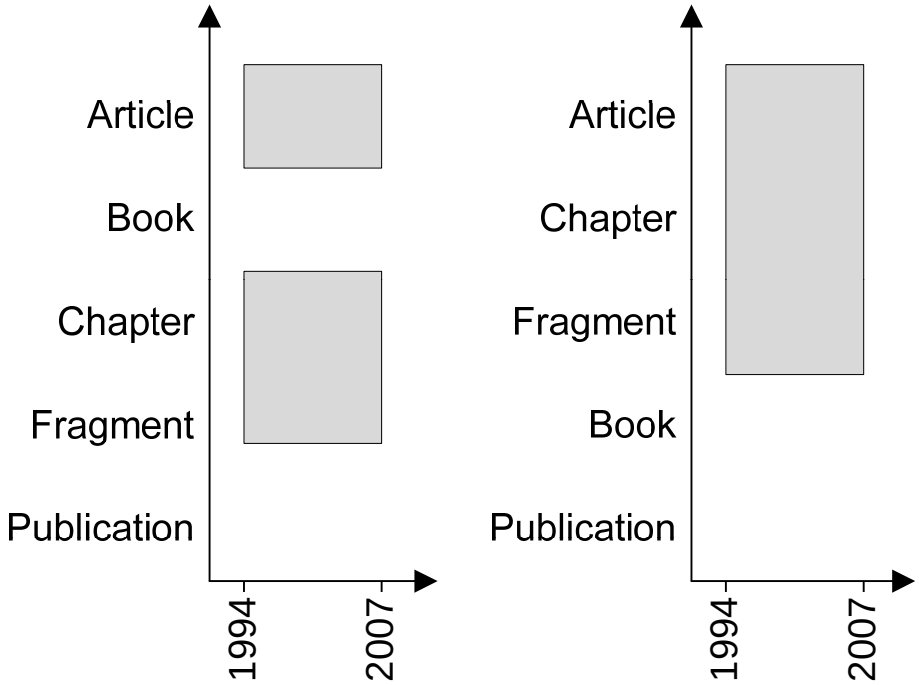
- Result of rake-contract heuristic



$q = 1$
$r = 3$

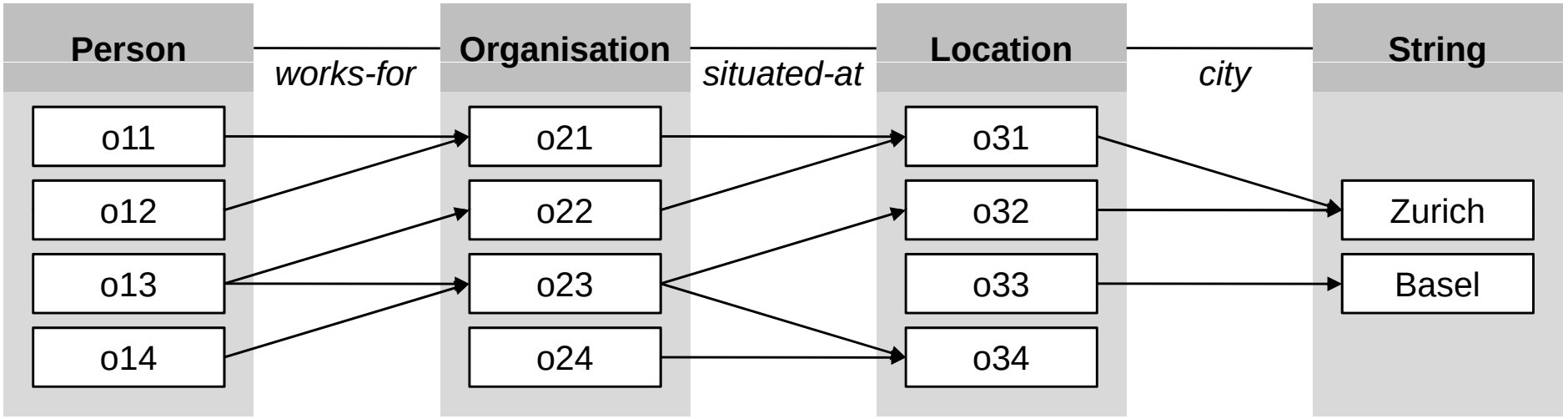
Multi-Key Type Index (MT-Index)

- Compromise between type and key grouping approaches
- Interprets type membership as an additional object attribute
  - symmetrical indexing of object types and attribute values
  - able to support indexing of more than one attribute with single search data structure
- Can be built using any multi-dimensional data structure
  - BV-tree, hB-tree or hB$^{\Pi}$-tree
- Performance of MT-index depends on linerisation of indexed type hierarchy



- Query evaluation
  - traversal to collect set of relevant disk page addresses
  - check all records and discard those not qualifying for request

Aggregation Path Indexing



- Backward queries without full object scans
  - find all persons working for an organisation in Zurich
- Forward without retrieving intermediate objects
  - find the city where person o12 works
- Path decomposition schemes

Multi-Index (MX)

- Introduced by GemStone in 1989
- Divide path of arbitrary length into sub-paths
  - sub-paths all have length one
  - index maintained over sub-paths
- Multi-Index for path *P*, *MX(P)*, consists of a set of index components $IX(P_i)$
  - MX(Person.works-for.situated-at.city) = {IX(Person.works-for), IX(Organisation.situated-at), IX(Location.city)}
- Query evaluation
  - concatenating *n* index edges require *n* index scans
  - supports backwards but not forward traversals and queries

Multi-Index Example

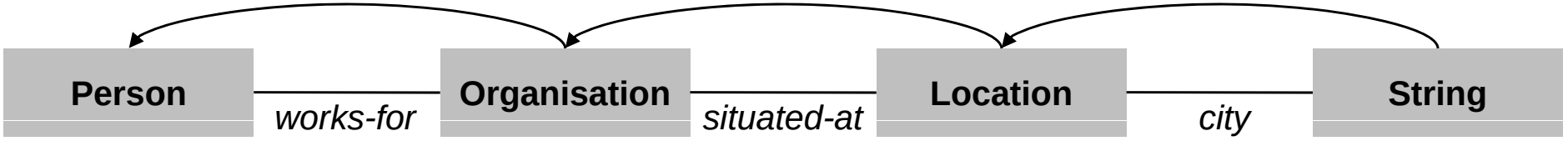
Multi-Index MX(Person.works-for.situated-at.city)

IX(Person.works-for)	
o21	o11 o12
o22	o13
o23	o13 o14

IX(Organisation.situated-at)	
o31	o21 o22
o32	o23
o34	o23 o24

IX(Location.city)	
Basel	o33
Zurich	o31 o32

Indexing graph for the Multi-Index



Access Support Relations (ASR)

- ASR for aggregation path of length *n* is *(n+1)*-ary relation
- Defined using binary ASRs for sub-paths of length one
  - compositions: ASR_{can}, ASR_{full}, ASR_{left} and ASR_{right}
  - decompositions: Nested Index, Path-Index and Join-Index

ASR(Person.works-for)	
o11	o21
o12	o21
o13	o22
o13	o23
o14	o23

ASR(Organisation.situated-at)	
o21	o31
o22	o31
o23	o32
o23	o34
o24	o34

ASR(Location.city)	
o31	Zurich
o32	Zurich
o33	Basel

ASR Compositions

ASR(Person.works-for) ⋈ ASR(Organisation.situated-at) ⋈ ASR(Location.city)

ASR_{can}(Person.works-for.situated-at.city)			
o11	o21	o31	Zurich
o12	o21	o31	Zurich
o13	o22	o31	Zurich
o13	o23	o32	Zurich
o14	o23	o32	Zurich

ASR_{full}(Person.works-for.situated-at.city)			
–	–	o33	Basel
–	o24	o34	–
o11	o21	o31	Zurich
o12	o21	o31	Zurich
o13	o22	o31	Zurich
o13	o23	o32	Zurich
o14	o23	o32	Zurich
o14	o23	o34	–

ASR(Person.works-for) ⋈ ASR(Organisation.situated-at) ⋈ ASR(Location.city)

ASR Compositions

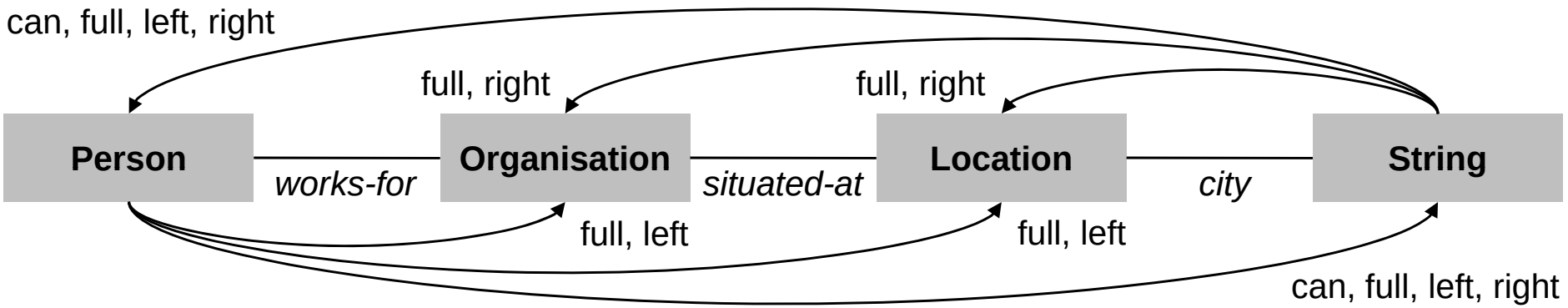
(ASR(Person.works-for) ⋈ ASR(Organisation.situated-at)) ⋈ ASR(Location.city)

ASR_{left}(Person.works-for.situated-at.city)			
o11	o21	o31	Zurich
o12	o21	o31	Zurich
o13	o22	o31	Zurich
o13	o23	o32	Zurich
o14	o23	o32	Zurich
o14	o23	o34	–

ASR_{right}(Person.works-for.situated-at.city)			
–	–	o33	Basel
o11	o21	o31	Zurich
o12	o21	o31	Zurich
o13	o22	o31	Zurich
o13	o23	o32	Zurich
o14	o23	o32	Zurich

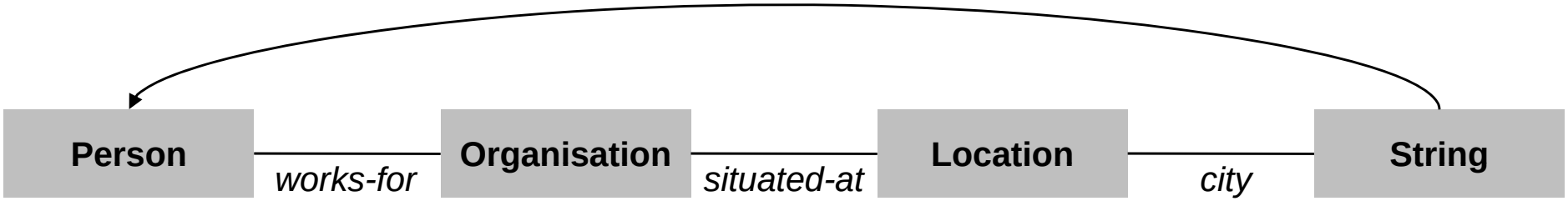
ASR(Person.works-for) ⋈ (ASR(Organisation.situated-at) ⋈ ASR(Location.city))

ASR Compositions Indexing Graph



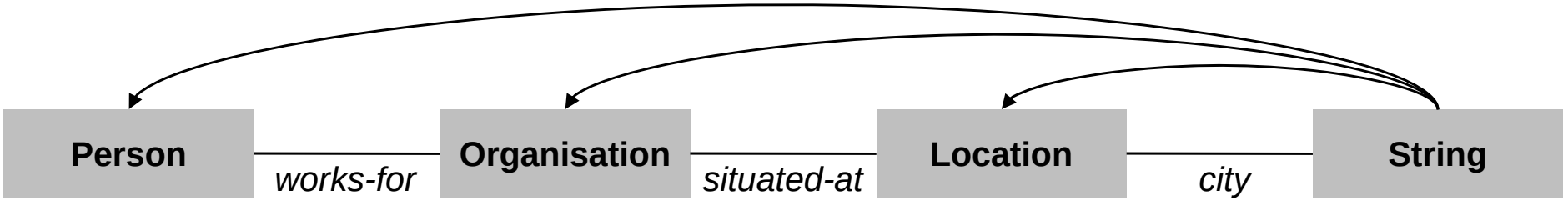
- Queries that do not traverse the path at either endpoint cannot be answered efficiently
- Aggregation path can be split into sub-paths
  - for each an ASR extension (partition) is maintained
  - the set of partitions is called a decomposition of an ASR

Nested Index (NX)



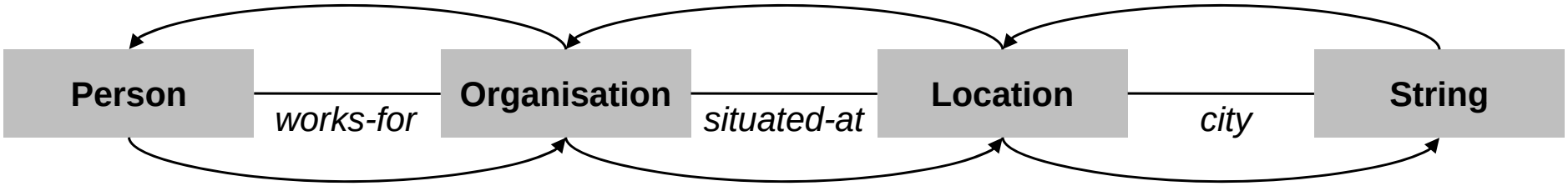
- Only allows backwards traversals of the full path
- Equivalent to backward traversal in ASR_{can}
- A Nested Index *NX(P)* for a path *P* with length one is equivalent to a Multi-Index *MX(P)* for the same path

Path-Index (PX)



- Equivalent to backwards traversal of right-complete ASR
- A Path-Index *PX(P)* for a path *P* with length one is equivalent to a Multi-Index *MX(P)* and a Nested Index *NX(P)* for the same path

Join-Index (JX)



- Originally introduced to optimise joins in relational DBMS
- Join-Index consists of a set of binary join indices
  - one binary join index per sub-path of length one of indexed path
  - each binary join index kept redundantly in two data structures
  - binary join index equivalent to corresponding binary ASR

Collection Operations

- Object-oriented databases allow multi-valued attributes
  - sets, bags, lists and arrays of values
  - new modelling features and enhanced expressiveness
  - increased complexity of indexing and query optimisation
- OQL provides constructors and operators for collections

```
select p.name from Publication p
where "XCM" in p.topics

select p.name from Publication p
where Set("Indexing", "Storage", "OQL") <= p.topics

select p.name from Publication p
where Set("Indexing", "Storage", "OQL") >= p.topics
```

Signature Files

- Originally proposed for information retrieval
- Index construction for a multi-valued property of a type
  - compute element signature for every possible attribute value
  - compute set signature based on element signatures
  - signature file stores set signatures for all objects
- Query evaluation over multi-valued properties
  - compare query set S_Q to each target S_T by matching the query signature $sig(S_Q)$ and the target signature $sig(S_T)$
  - matching of signatures yields drops
    S_T is a drop for $S_Q \subseteq S_T$ iff $(sig(S_Q) \wedge sig(S_T)) = sig(S_Q)$
    S_T is a drop for $S_Q \supseteq S_T$ iff $(sig(S_Q) \wedge sig(S_T)) = sig(S_T)$
  - drops can be actual drops or false drops

Signature Files Example

Query Set S_Q

Element	Signature
Indexing	00100001
Storage	01000001
OQL	00100010
Query	**01100011**

$S_Q \subseteq S_T$

Target Sets S_T

Element	Signature
Indexing	00100001
Storage	01000001
OQL	00100010
Modelling	10010000
Target	**11110011**

actual drop

Element	Signature
Storage	01000001
OQL	00100010
SQL	01100000
Recovery	10000100
Target	**11100111**

false drop

$S_Q \supseteq S_T$

Element	Signature
Storage	01000001
OQL	00100010
Target	**01100011**

actual drop

Element	Signature
Storage	01000001
SQL	01100000
Target	**01100001**

false drop

Literature

- Thomas A. Mueck and Martin L. Polaschek: **Index Data Structures in Object-Oriented Databases**, *Kluwer Academic Publishers 1997*
- Elisa Bertino, Beng Chin Ooi, Ron Sacks-Davis, Kian-Lee Tan, Justin Zobel, Boris Shindlovsky and Barbara Catania: **Indexing Techniques for Advanced Database Systems**, *Kluwer Academic Publishers 1997*

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Next Week

Version Models

- Temporal Databases
- Engineering Databases
- Software Configuration Systems