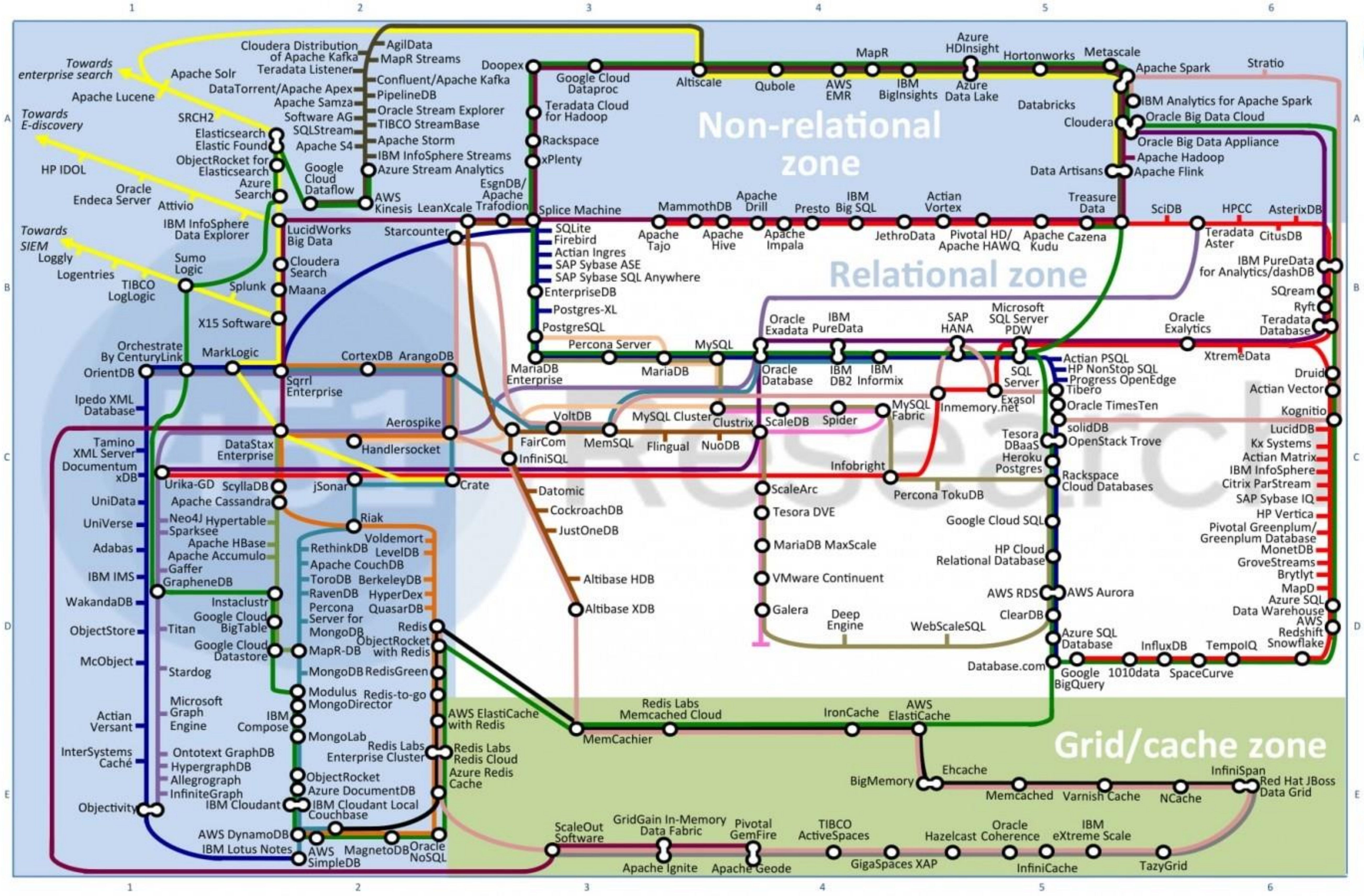
# Physical Design for Document Stores

**Moditha Hewasinghage**

Advisors : Alberto Abelló , Jovan Varga (UPC)
Esteban Zimányi (ULB)

Nineth European Eighth European Business Intelligence & Big Data
Summer School (eBISS 2019) Berlin, Germany



# 451 Research

# Data Platforms Map

January 2016

Key:

- General purpose
- Specialist analytic
- -as-a-Service
- BigTables
- Graph
- Document
- Key value stores
- Key value direct access
- Hadoop
- MySQL ecosystem
- Advanced clustering/sharding
- New SQL databases
- Data caching
- Data grid
- Search
- Appliances
- In-memory
- Stream processing

https://451research.com/state-of-the-database-landscape

# What is the optimal data design for a given dataset and a query load?

01 Which data store ?

02 Which design ?

The design affects performance [Atzeni,2016]

The decision is not trivial

# Benefits of Finding the Optimal Design(s)

- More control over the data design as opposed to trial and error
- Improved data quality
- Avoid costly operations occurring from poor design
- Reduce data movement cost
- Reduce redesign cost

# Data Design Problem

## Alternative data models

- Relational
- Document
- Key-Value
- Wide column
- Graph

## Alternative Designs

- Structures
- Collections
- Nesting

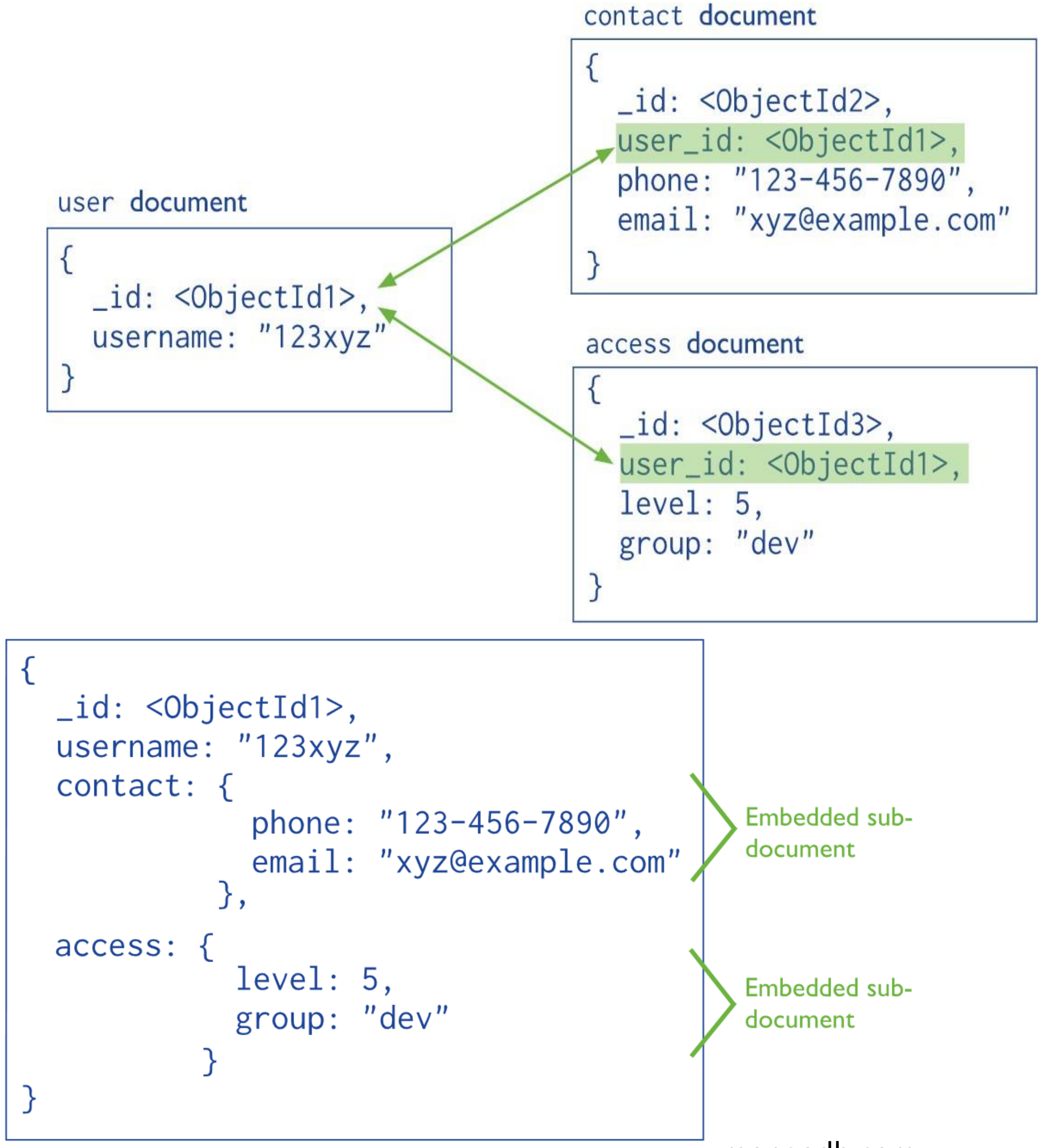
## Affects several criteria

- Storage space
- Query cost
- Execution time
- Degree of heterogeneity

# Multicriteria Optimization for Data Design

- Considers several alternative solutions
- Contradicting requirements
- Find the optimal or set of  Pareto-optimal solutions
- Applied in different domains

# Initial Focus on Document Stores



user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
           },
  access: {
             level: 5,
             group: "dev"
          }
}
```

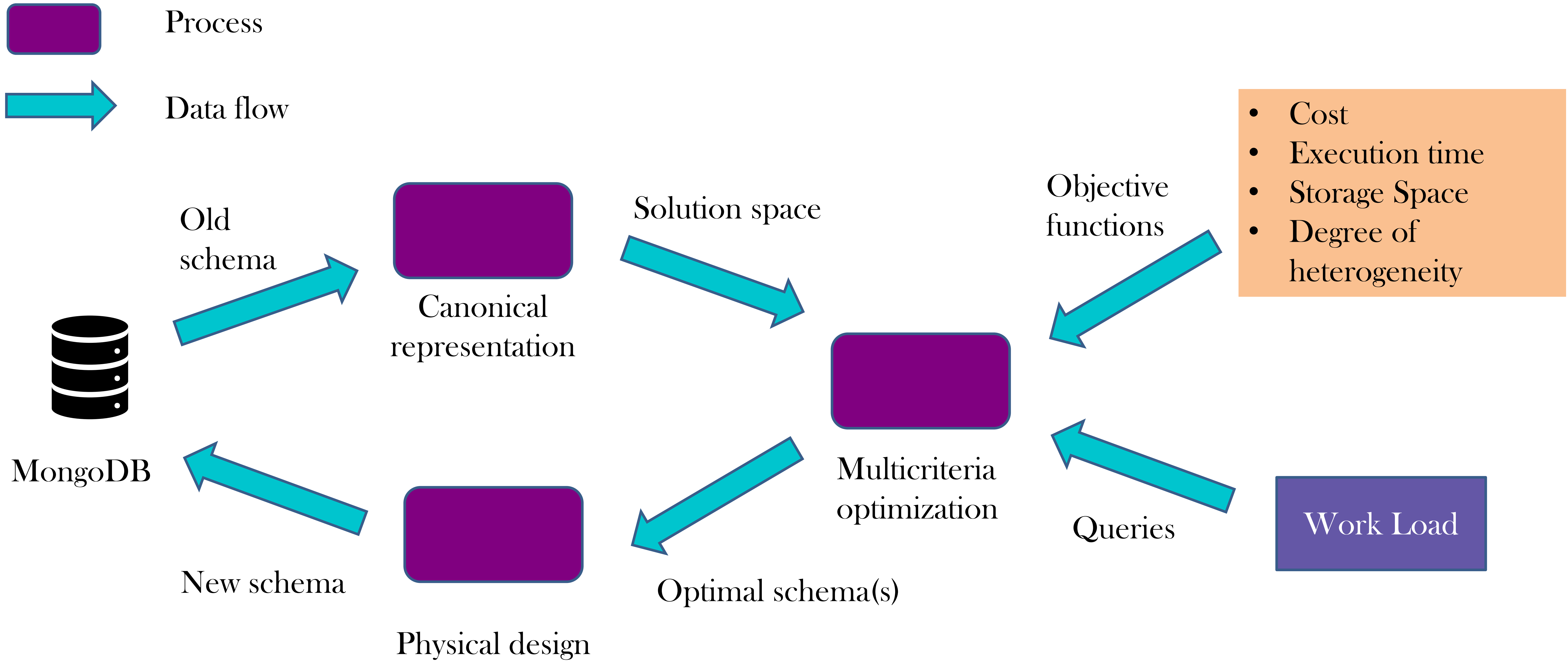
www.mongodb.com

Has multiple characteristics

- Semi-structured
- Heterogeneous collections
- Allows nesting

Later extend to other data stores

# Workflow of the Approach



Process

Data flow

MongoDB

Old schema

Canonical representation

Solution space

Multicriteria optimization

Objective functions

- Cost
- Execution time
- Storage Space
- Degree of heterogeneity

Queries

Work Load

Optimal schema(s)

Physical design

New schema

# Search Space

Alternative data designs

Design dimensions

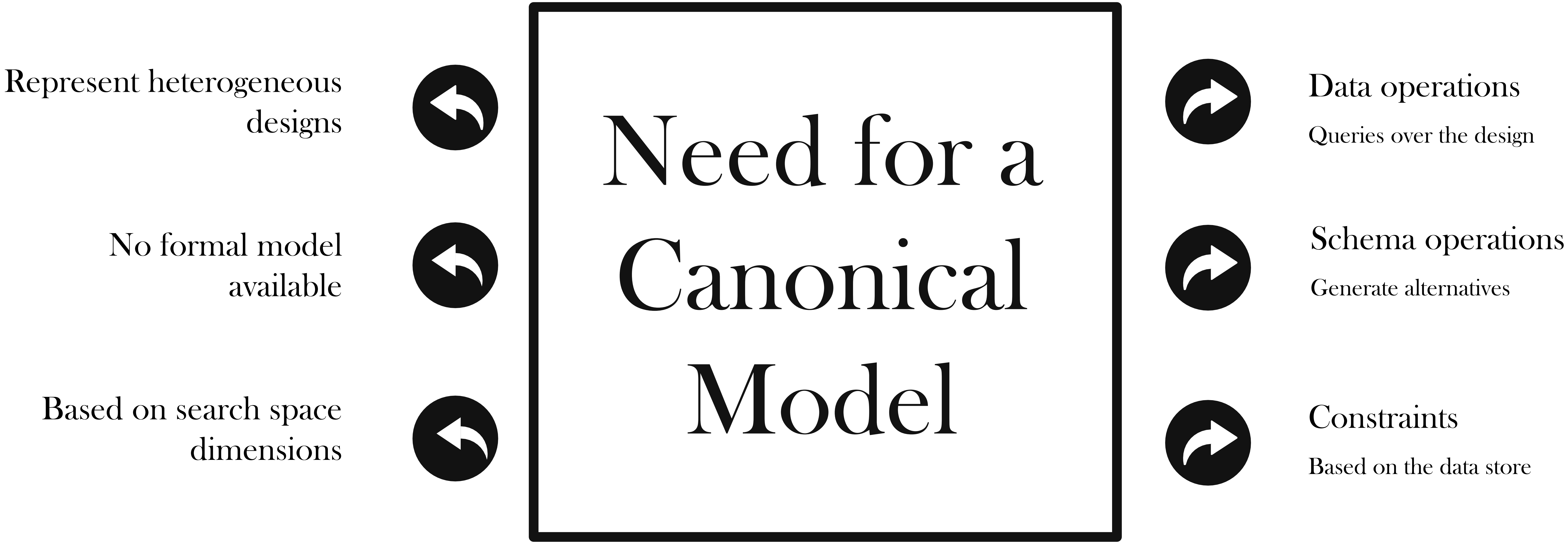
- Structuring
- Grouping
- Nesting

Based on existing work

- SOS Model [1]
- NoAM [2]

[1] P. Atzeni, F. Bugiotti, and L. Rossi. Uniform access to NoSQL systems. Information Systems, 43, 2014.

[2] F. Bugiotti, L. Cabibbo, P. Atzeni, and R. Torlone. Database design for NoSQL systems. In International Conference on Conceptual Modeling. Springer, 2014



# Need for a Canonical Model

Represent heterogeneous designs

No formal model available

Based on search space dimensions

Data operations

Queries over the design

Schema operations

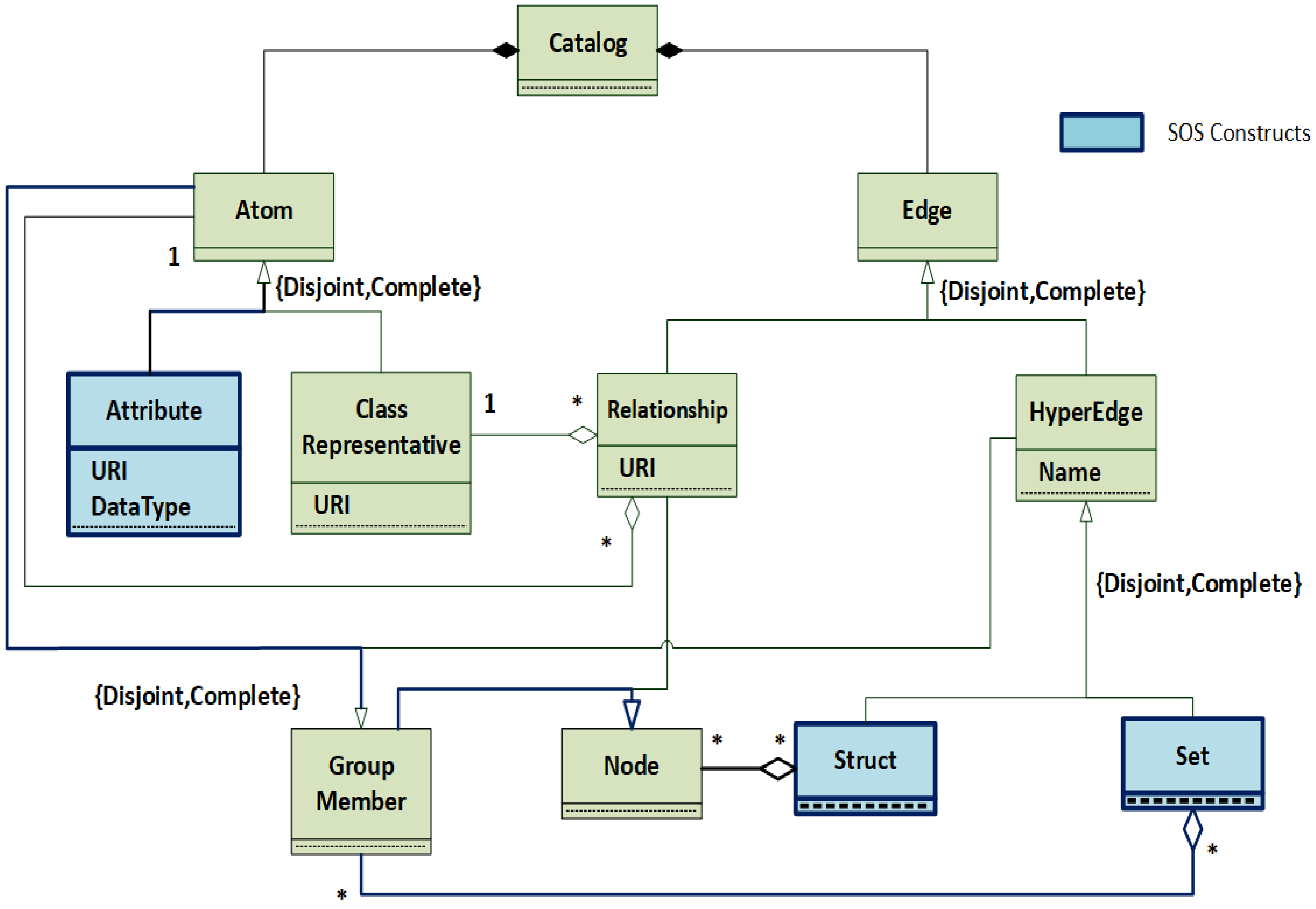
Generate alternatives

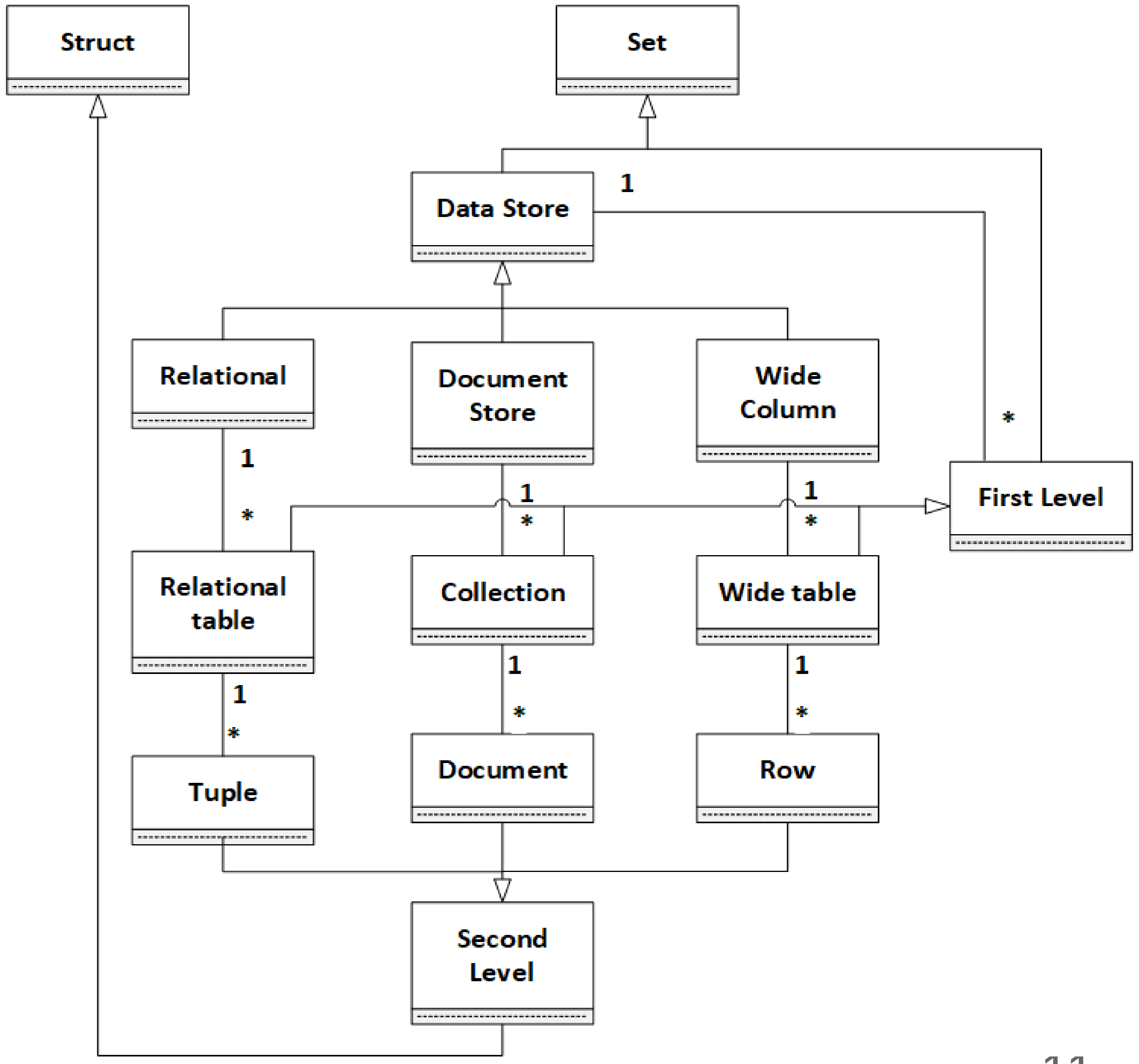
Constraints

Based on the data store

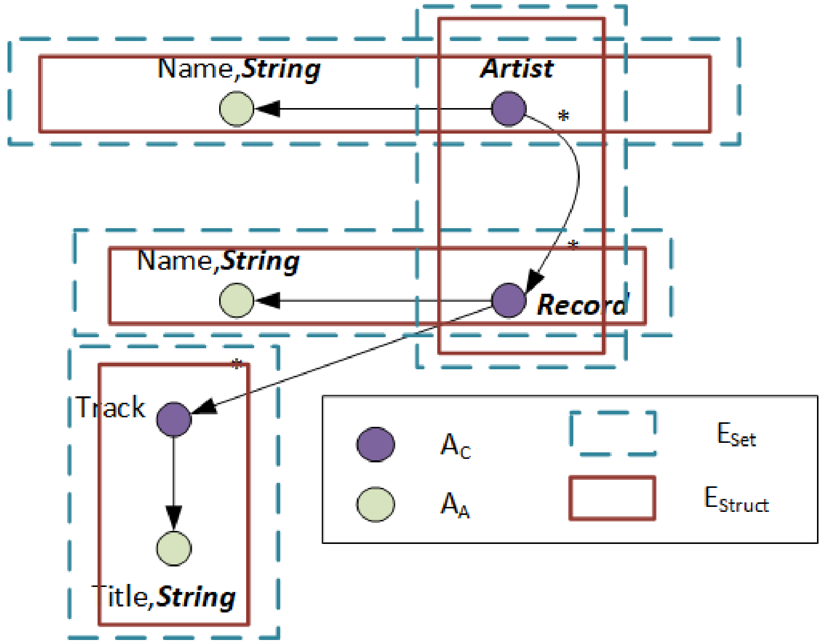
# A hypergraph-based Canonical Design Model

M. Hewasinghage, J. Varga, A. Abelló, and E. Zimányi. Managing Polyglot Systems Metadata with Hypergraphs.
In International Conference on Conceptual Modeling. ER, 2018.

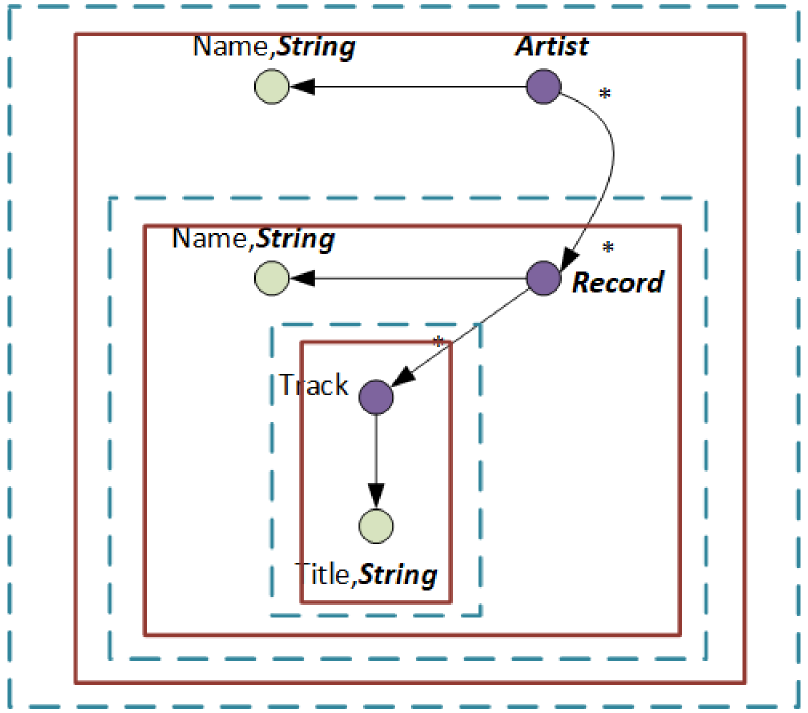




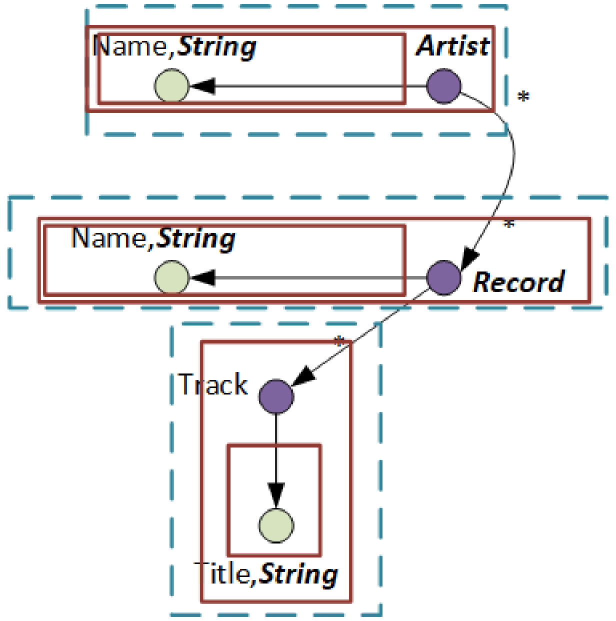
# A hypergraph-based Canonical Design Model Cont.



Relational



Document Store



Column Family

- RDF exemplars in a graph
- Build generalized hypergraph representing different design constructs
- Represent heterogeneous data models
  - Relational
  - Document Store
  - Column Family
- Identified constraints over different data models
- Simple query generation over the design
- Schema operations in the solution space for alternative designs (transformations)
- Modify the query algorithm to calculate other measures (size, frequency, runtime)

https://github.com/modithah/ESTOCADA-CATALOG

# Objective Functions

## Storage Space

- Calculated from the design
- Affected by
  - Number of collections
  - Nesting
  - Number of objects

## Query Cost

- Disk I/O
- Memory usage

## Execution Time

- Parallelism

## Degree of Heterogeneity

- Within objects in the collection

# Cost Model for Queries in Document Stores

M. Hewasinghage, A. Abelló, J. Varga, and E. Zimányi. In International Conference on Data Engineering (ICDE) 2020 (Under review).

## No existing cost model

- Primitive approaches for query processing

## Based on disk I/O

- Similar to RDBMS
- Extended by including memory management

## MongoDB and Couchbase

- Different cache policies

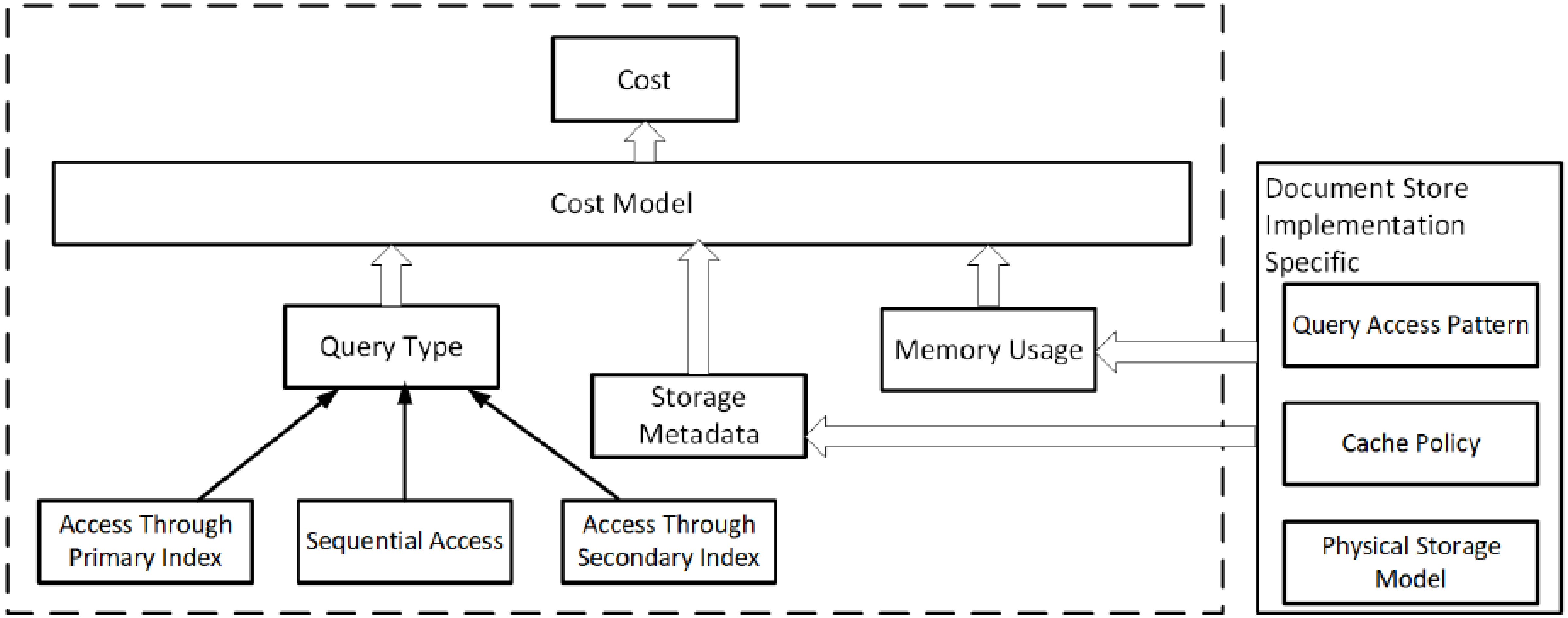
## Random access queries

- Access a document by primary identifier

## Key parameters

- Document size
- Number of documents
- Probability of access
- Memory size

# Overview of the Cost Model



# Generic Cost Model

| | |
|---|---|
| $M$ | Total memory available for the document store |
| $Bsize_d$ | Block size for data |
| $Bsize_i$ | Block size for index |
| $T_m$ | Time to read a block from cache |
| $T_d$ | Time to read a block from disk |
| $C$ | A collection |
| $Size_d(C)$ | Average document size of a collection |
| $Size_i(C)$ | Average index entry size of a collection |
| $\|C\|$ | Number of documents in a collections |
| $R_d(C)$ | Average number of documents in a block |
| $R_i(C)$ | Average number of index entries in a block |
| $B_d(C)$ | Total document size in blocks |
| $B_i(C)$ | Total index entry size in blocks |

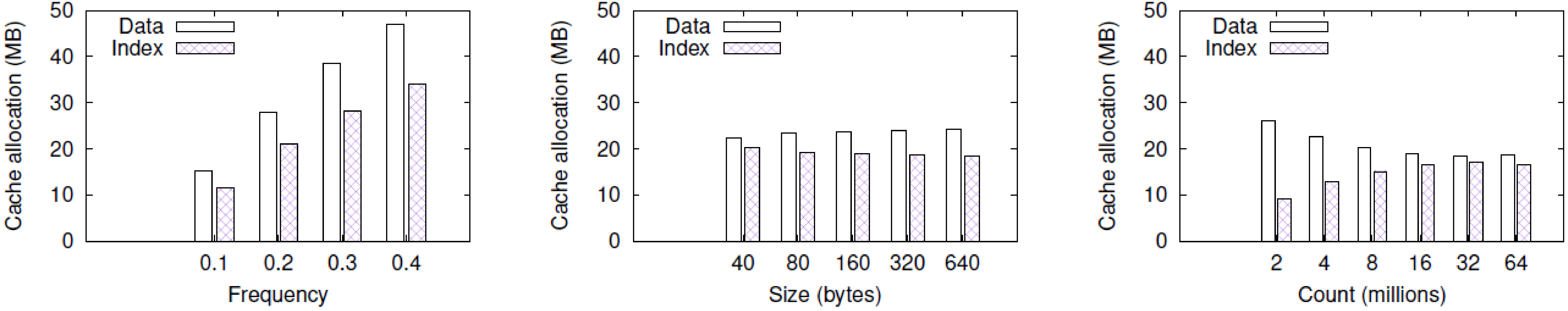
$$R_d(C) = f.\left\lceil \frac{Bsize_d}{Size_d(C)} \right\rceil \qquad R_i(C) = f.\left\lceil \frac{Bsize_i}{Size_i(C)} \right\rceil$$

$$B_d(C) = \left\lceil \frac{|C|}{R_d(C)} \right\rceil \qquad B_i(C) = \left\lceil \frac{|C|}{R_i(C)} \right\rceil$$

$$P_d(C) = \frac{M_d(C)}{B_d(C)} \qquad P_i(C) = \frac{M_i(C)}{B_i(C)}$$

$$\begin{aligned} Cost_{Rand} &= \frac{T_m * P_i(C) + T_d * (1 - P_i(C))}{2} \\ &+ \frac{T_m * P_d(C) + T_d * (1 - P_d(C))}{2} \\ &= \frac{T_m * (P_i(C) + P_d(C))}{2} + \frac{T_d * (2 - (P_i(C) + P_d(C)))}{2} \end{aligned}$$

# Parameters Affecting the Memory Usage

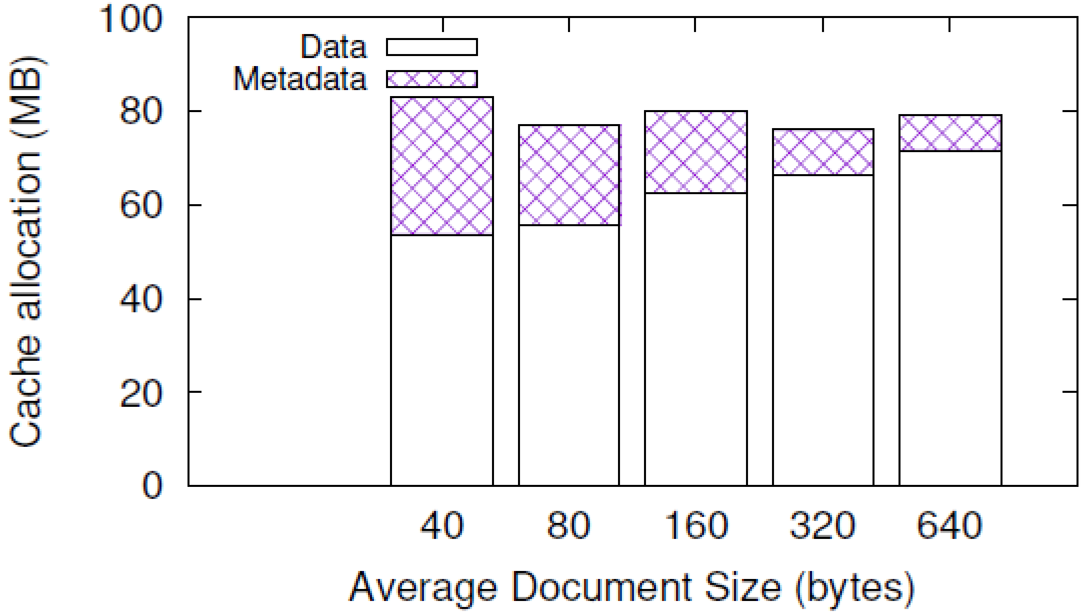


(a) Access frequency

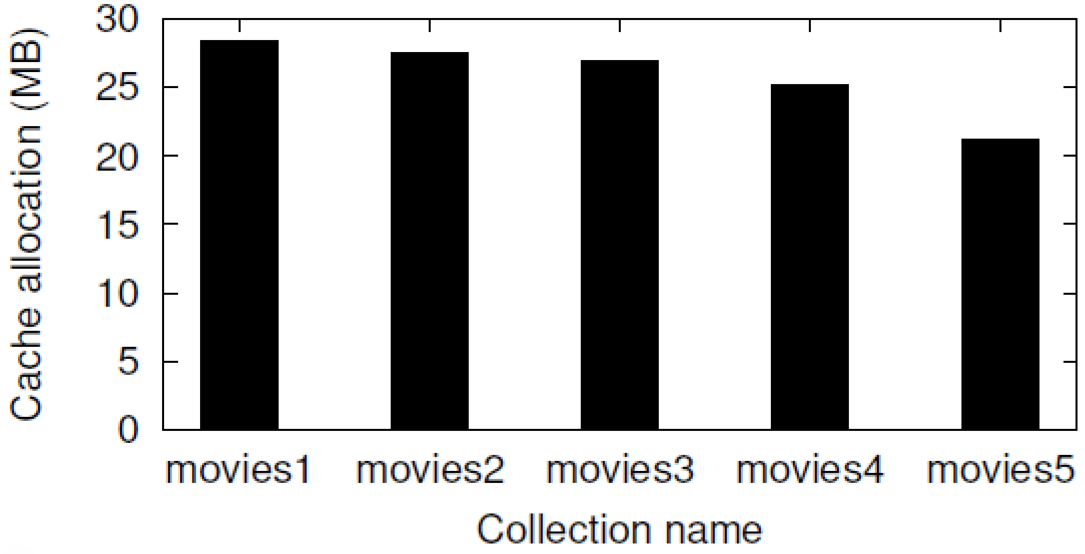
(b) Document size

(c) Document count

# Memory Usage Estimation



Memory utilization in Couchbase Server

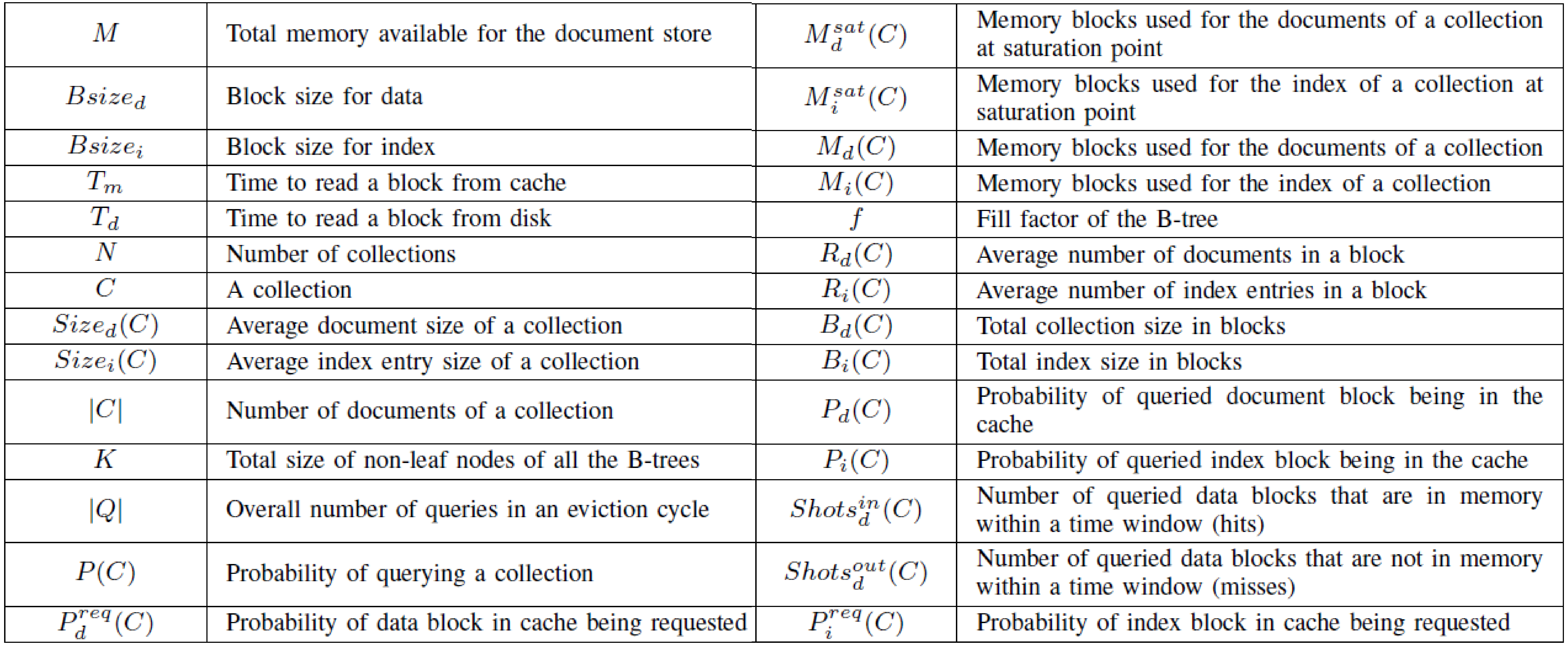


MongoDB cache policy prioritizing the collection name

- Predefined memory size in Couchbase (buckets)
  - Keep metadata in memory
  - Full eviction

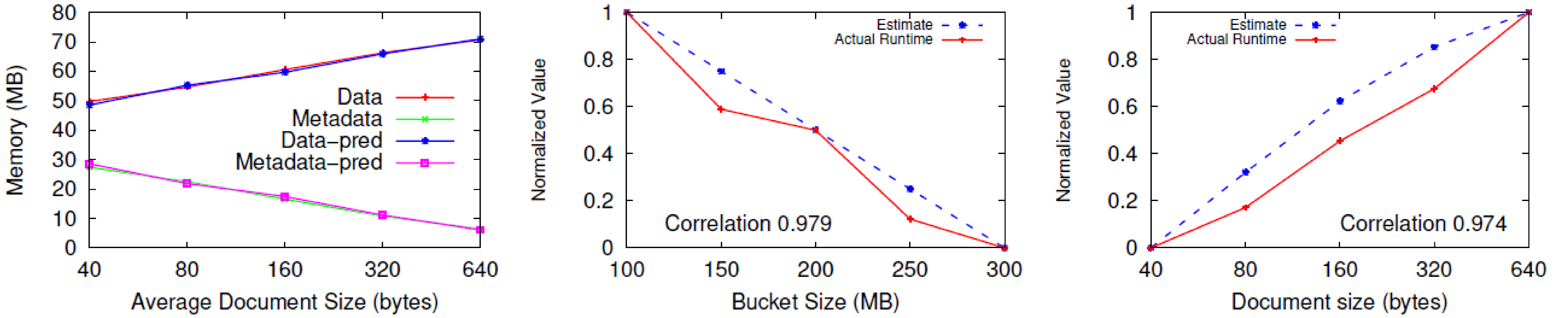
- LRU-like cache policy in MongoDB
  - Cache policy biased towards collection name (https://jira.mongodb.org/browse/WT-4732)
  - Isolated the issue, made fixes, notified developers
  - Fix will be released in WiredTiger 3.2.1

# Cost Model Parameters



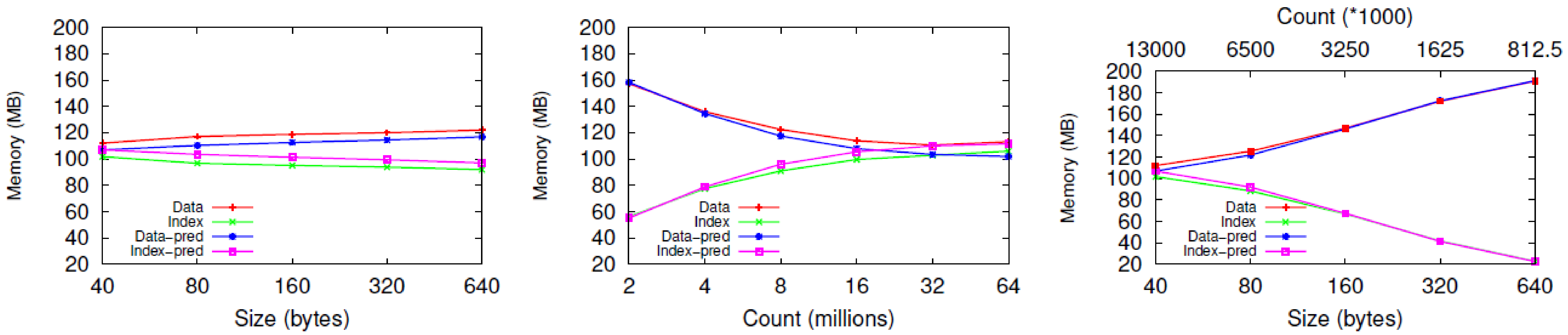
| | | | |
|---|---|---|---|
| $M$ | Total memory available for the document store | $M_d^{sat}(C)$ | Memory blocks used for the documents of a collection at saturation point |
| $Bsize_d$ | Block size for data | $M_i^{sat}(C)$ | Memory blocks used for the index of a collection at saturation point |
| $Bsize_i$ | Block size for index | $M_d(C)$ | Memory blocks used for the documents of a collection |
| $T_m$ | Time to read a block from cache | $M_i(C)$ | Memory blocks used for the index of a collection |
| $T_d$ | Time to read a block from disk | $f$ | Fill factor of the B-tree |
| $N$ | Number of collections | $R_d(C)$ | Average number of documents in a block |
| $C$ | A collection | $R_i(C)$ | Average number of index entries in a block |
| $Size_d(C)$ | Average document size of a collection | $B_d(C)$ | Total collection size in blocks |
| $Size_i(C)$ | Average index entry size of a collection | $B_i(C)$ | Total index size in blocks |
| $\|C\|$ | Number of documents of a collection | $P_d(C)$ | Probability of queried document block being in the cache |
| $K$ | Total size of non-leaf nodes of all the B-trees | $P_i(C)$ | Probability of queried index block being in the cache |
| $\|Q\|$ | Overall number of queries in an eviction cycle | $Shots_d^{in}(C)$ | Number of queried data blocks that are in memory within a time window (hits) |
| $P(C)$ | Probability of querying a collection | $Shots_d^{out}(C)$ | Number of queried data blocks that are not in memory within a time window (misses) |
| $P_d^{req}(C)$ | Probability of data block in cache being requested | $P_i^{req}(C)$ | Probability of index block in cache being requested |

# Results Obtained – Couchbase



(a) Couchbase cache usage prediction (b) Time estimate with different bucket size (c) Time estimate with different document size

# Results Obtained – MongoDB

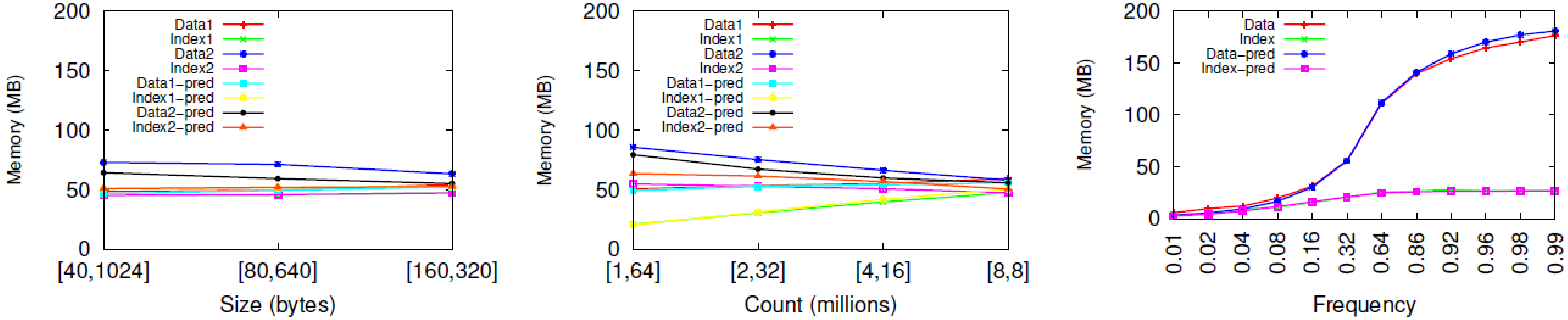


(a) Document size

(b) Document count

(c) Size and count

**Single Collection**



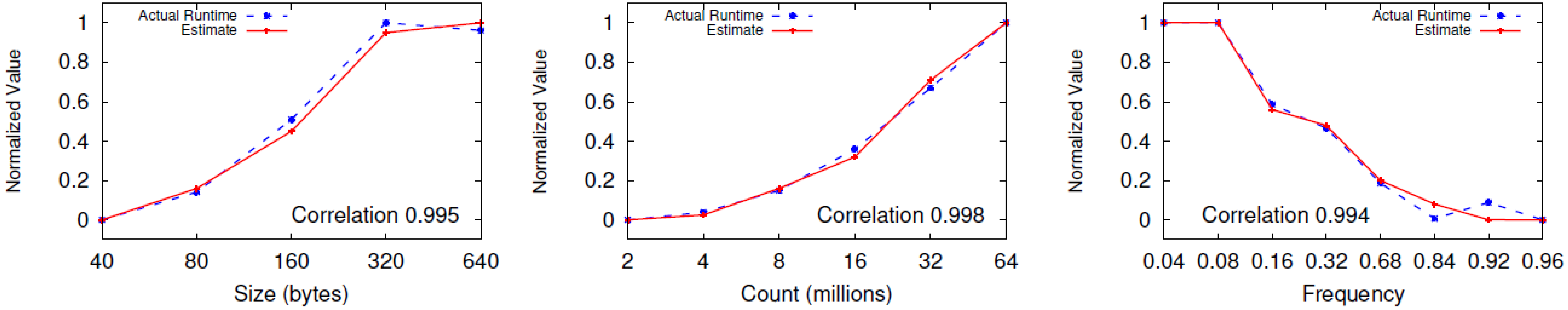
(a) Document size

(b) Document count

(c) Access frequency

**Two Collections**

# Runtime Estimates - MongoDB



(a) Document size (b) Document Count (c) Access frequency

- Cost model for document stores using disk I/O, memory, storage size (size, count), and access frequency.
- Predicting the memory usage with average error of 3% for Couchbase, 9% for MongoDB
- Accurately predict the time estimation trend with high correlation (0.97 Couchbase, 0.99 MongoDB )

# Publication Strategy

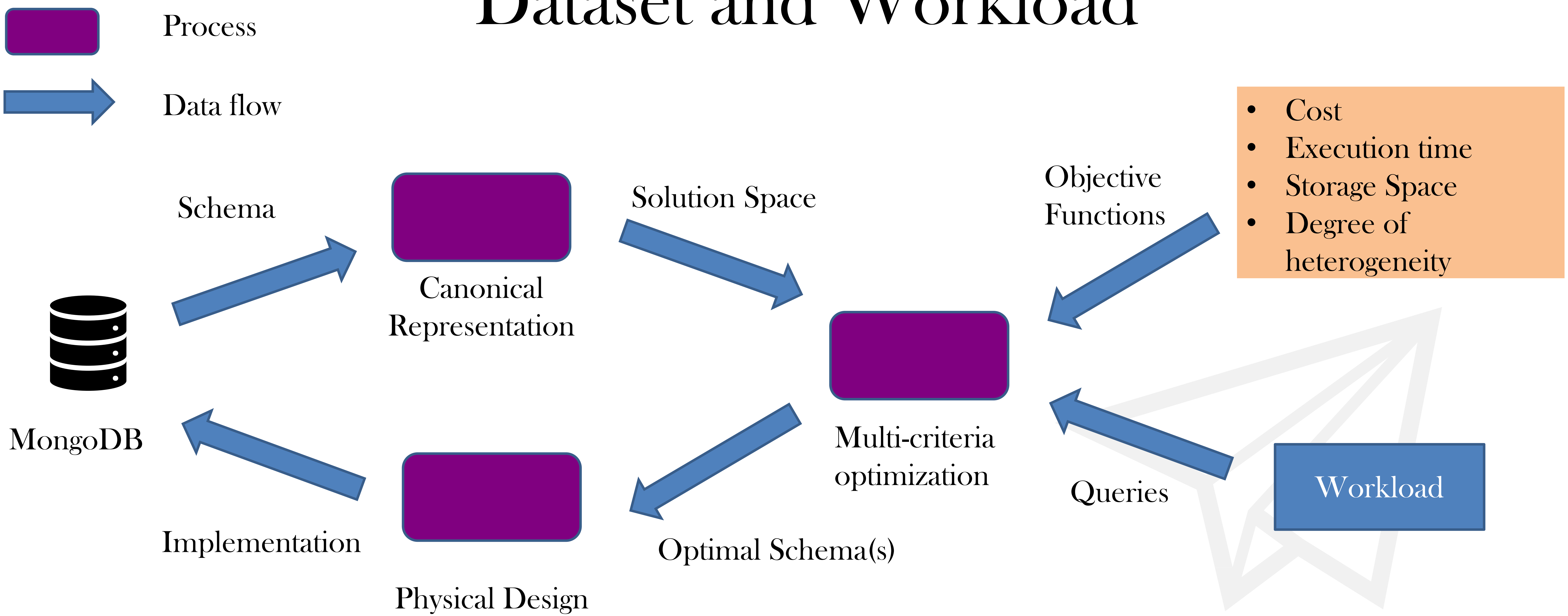
- Managing Polyglot Systems Metadata with Hypergraphs (Published ER 2018)
- A Cost Model for Queries in Document Stores (Under review ICDE 2020)
- Cost-based Data Design for Document Stores (DASFAA 2020)
  - Extend the algorithms for cost parameters
    - Storage size
    - Query cost
    - Access probability
  - Evaluate pre-defined designs to choose the best one
- A Framework for Optimal Data Design in Document Stores (SIGMOD Demo – Jan 2020)
- Multi-criteria Decision Making in Data Design for Document Stores (TKDE – March 2020)
  - Extend the hypergraph model transformations
  - Implement multicriteria based data design framework
- Data Design for NoSQL Systems (ICDE 2021 – July 2020)
  - Extend to another data store

# Planned PhD Courses

| Activity | ECTS |
|---|---|
| Spanish Language (A1, A2) | 2.5 |
| 4th International Winter School on Big Data | 2 |
| Big Data analytics | 2 |
| Semantic Data Management (SDM) | 2 |
| Seminar on Crypto currency | 1 |
| Seminar on Text Analytics | 1 |
| French Language (B1) | 2.5 |
| IT4BI Summer School | 2 |
| Project management for PhD candidates | 1 |
| Dutch-Belgian Database Day | 1 |
| Academic Writing | 2 |
| Academic Communication | 2 |
| ER 2018 Participation | 2 |
| IT4BI-DC Doctoral Colloquium | 3 |
| Research group seminar | 2 |
| Offered seminar | 1 |
| Offered seminar | 1 |
| Offered seminar | 1 |
| Conference Presentation | 2 |

26/30 Completed
14 Project
10 General
2 Project – Informal activity

# Optimal Design for a Given Dataset and Workload



Process

Data flow

MongoDB

Schema

Canonical Representation

Solution Space

Multi-criteria optimization

Objective Functions

- Cost
- Execution time
- Storage Space
- Degree of heterogeneity

Queries

Workload

Optimal Schema(s)

Physical Design

Implementation

# Constraints & Rules – Document Store

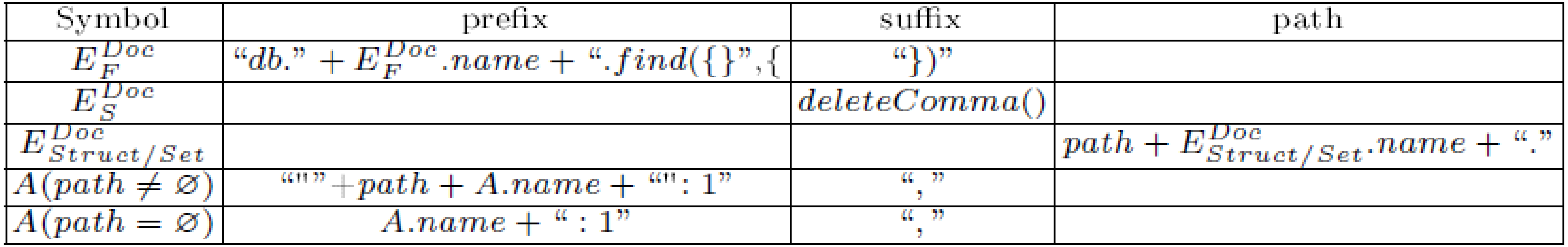
$$E_D^{Doc} \Rightarrow E_F^{Doc} *$$

$$E_F^{Doc} \Rightarrow E_S^{Doc} +$$

$$E_S^{Doc} \Rightarrow A_C\left(A\middle|E_{Set}^{Doc}\middle|E_{Struct}^{Doc}\right) *$$

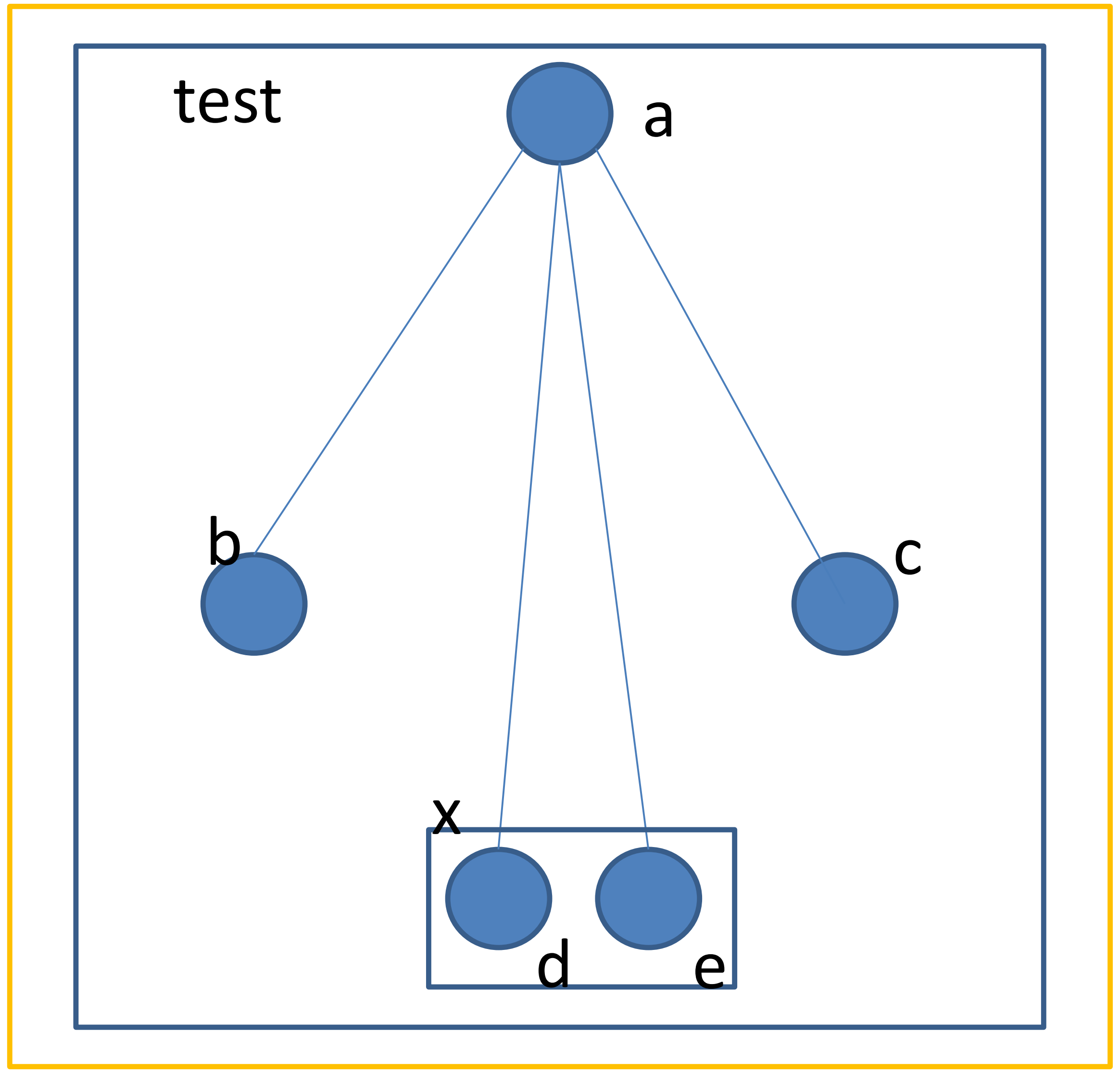
$$E_{Set}^{Doc} \Rightarrow E_{Struct}^{Doc} +$$

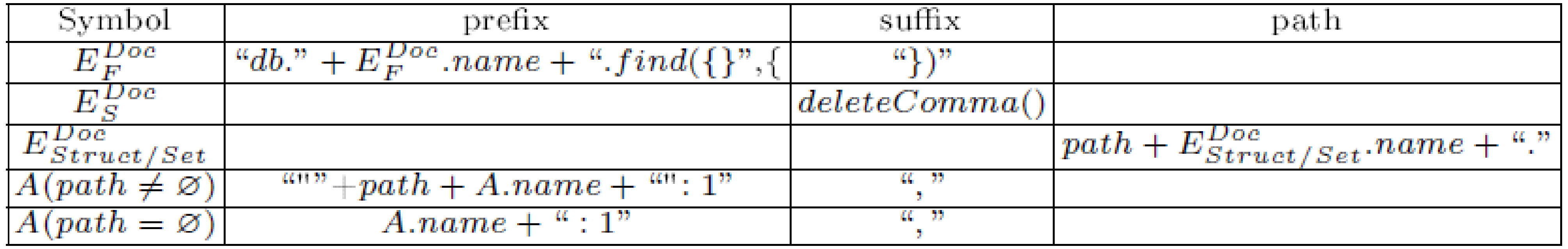
$$E_{Struct}^{Doc} \Rightarrow \left(A\middle|E_{Set}^{Doc}\middle|E_{Struct}^{Doc}\right) +$$



| Symbol | prefix | suffix | path |
|---|---|---|---|
| $E_F^{Doc}$ | $“db.” + E_F^{Doc}.name + “.find(\{\}”,\{$ | $“\})”$ | |
| $E_S^{Doc}$ | | $deleteComma()$ | |
| $E_{Struct/Set}^{Doc}$ | | | $path + E_{Struct/Set}^{Doc}.name + “.”$ |
| $A(path \neq \varnothing)$ | $“"” + path + A.name + “": 1”$ | “,” | |
| $A(path = \varnothing)$ | $A.name + “ : 1”$ | “,” | |

# Query generation on hypergraph





| Symbol | prefix | suffix | path |
|---|---|---|---|
| $E_F^{Doc}$ | “db.” $+ E_F^{Doc}.name +$ “$.find(\{\}$”,{ | “})” | |
| $E_S^{Doc}$ | | $deleteComma()$ | |
| $E_{Struct/Set}^{Doc}$ | | | $path + E_{Struct/Set}^{Doc}.name +$ “.” |
| $A(path \neq \varnothing)$ | “"” $+ path + A.name +$ “": 1” | “, ” | |
| $A(path = \varnothing)$ | $A.name +$ “ : 1” | “, ” | |

```
db.test.find({},{ a:1,b:1,"x.d":1})
```

```
{       a:””,
        b:””,
        c:””,
        x:{d:””,e:””}}
```

# Definitions

1. A polyglot catalog $C =< \boldsymbol{A}, \boldsymbol{E} >$ is a generalized hypergraph where A is a set of atoms and E is a set of edges.
2. The set of all atoms A is composed of two disjoint subsets of class atoms $A_C$ and attribute atoms $A_A$.

   $$A = A_C \mid A_A$$

3. The set of all edges E composed of two disjoints subsets of relationships $E_R$ that denote the connectivity between A, and hyperedges $E_H$ that denotes connectivity between o their constructs of C.

   $$E = E_R \mid E_H$$

4. A relationship $E_R^{x,y}$ is a binary edge between two atoms $A_x$ and $A_y$ and a URI u that represents the semantics of $E_R$. At least one of the atoms in the relationship must be an $A_C$.

   $$E_R^{x,y} =< A^x, A^y, u > |A^x, A^y \in \boldsymbol{A} \ \wedge \ (A^x \in \boldsymbol{A}_c \ \vee A^y \in \boldsymbol{A}_c)$$

# Definitions

5. The transitive closure of an edge E is denoted as $E^+$, where

$$E \in E^+, \forall e \in E^+: e \in e'.incidenceSet \Rightarrow e' \in E^+$$

6. A hyperedge $E_H$ is a subset of atoms $\boldsymbol{A}$ and edges $\boldsymbol{E}$ and it cannot be transitively contained in itself

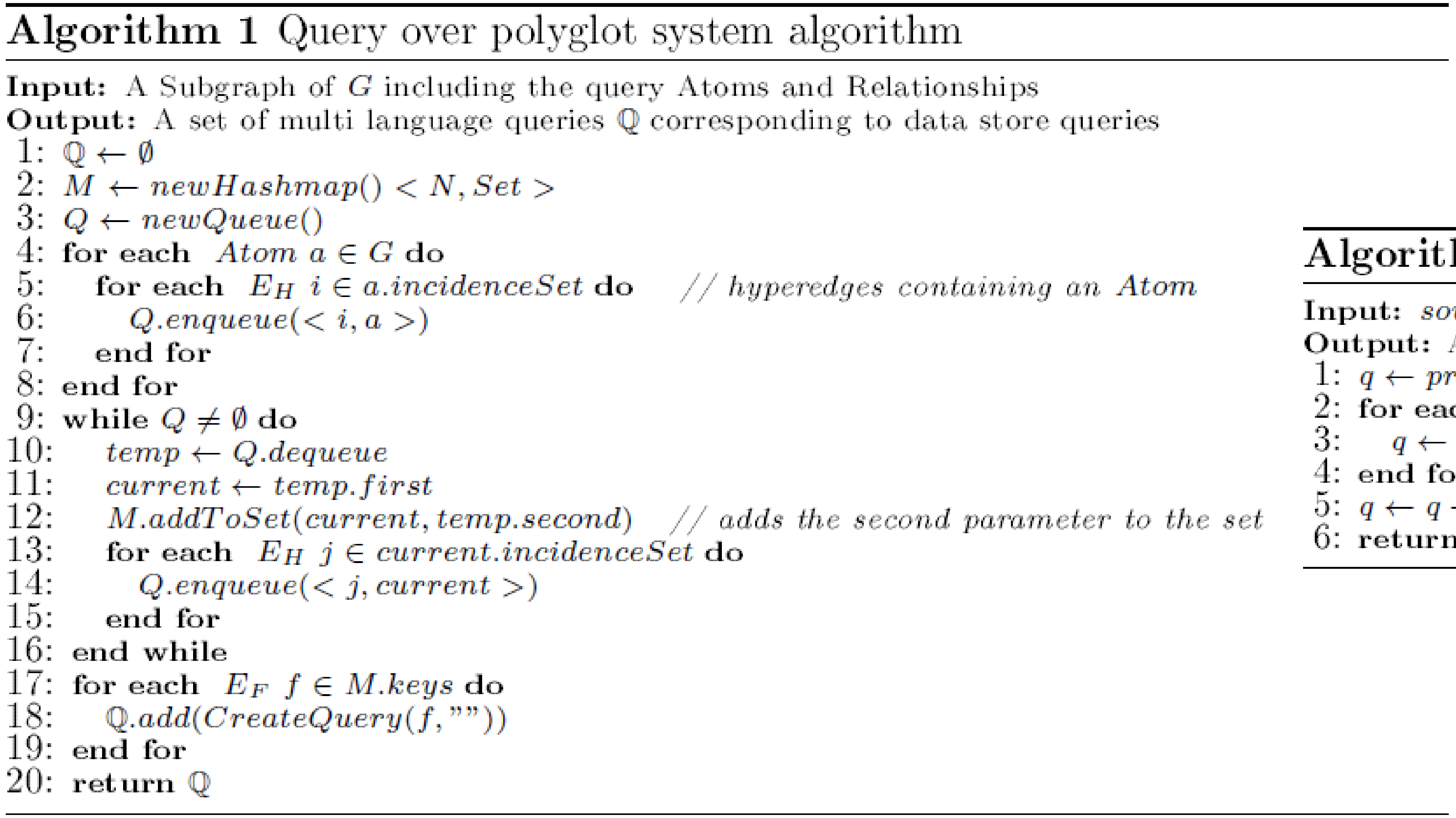
$$E_H \subseteq \boldsymbol{A} \cup \boldsymbol{E} \wedge E_H.incidenceSet \cap E_H^+ = \emptyset$$

7. A struct $E_{Struct}$ is a hyperedge that contains a set of atoms $\boldsymbol{A}$, relationships $\boldsymbol{E_R}$**,** and or hyperedges $\boldsymbol{E_H}$**.** All atoms within $E^+_{Struct}$must be connected by a set of $E_R$ that also belong to $E^+_{Struct}$

$$E_{Struct} \subseteq \boldsymbol{E_H} \cup \boldsymbol{A} \cup \boldsymbol{E_R} | \forall A_x, A_y \in E^+_{Struct}: \exists\{E_R^{x,x_1}, E_R^{x_1,x_2}, \dots, E_R^{x_n,y}\} \in \boldsymbol{E}^+_{Struct}$$

8. A Set $E_{Set}$ is a hyperedge that contains a set of arbitrary hyperedges $\boldsymbol{E_H}$ or/and atoms $\boldsymbol{A}$**.**

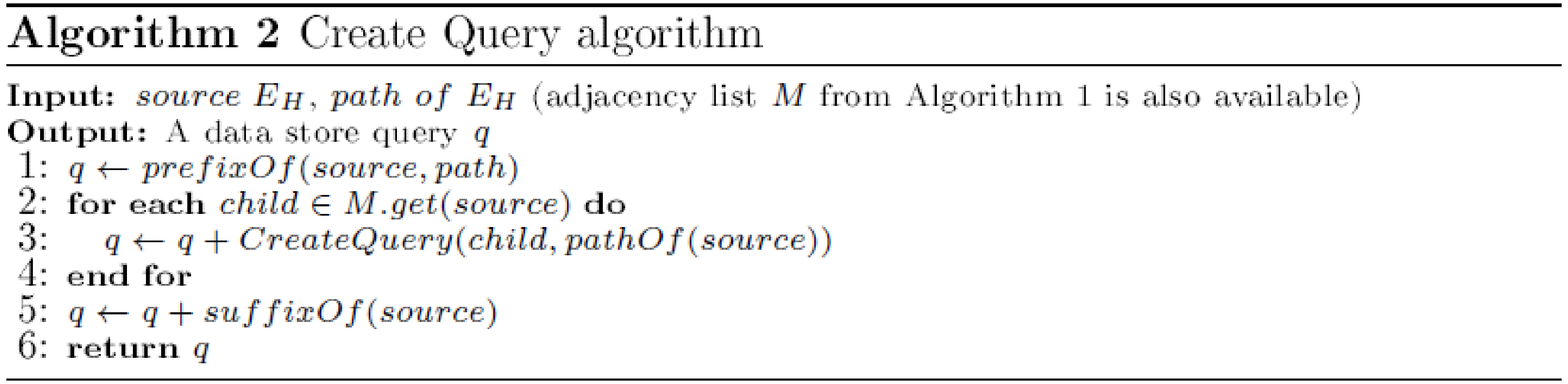
$$E_{Set} \subseteq \boldsymbol{E_H} \cup \boldsymbol{A}$$



**Algorithm 1** Query over polyglot system algorithm

**Input:** A Subgraph of $G$ including the query Atoms and Relationships
**Output:** A set of multi language queries $\mathbb{Q}$ corresponding to data store queries

```
$\mathbb{Q} \leftarrow \emptyset$
$M \leftarrow newHashmap() < N, Set >$
$Q \leftarrow newQueue()$
for each $Atom\ a \in G$ do
    for each $E_H\ i \in a.incidenceSet$ do    // hyperedges containing an Atom
        $Q.enqueue(< i, a >)$
    end for
end for
while $Q \neq \emptyset$ do
    $temp \leftarrow Q.dequeue$
    $current \leftarrow temp.first$
    $M.addToSet(current, temp.second)$    // adds the second parameter to the set
    for each $E_H\ j \in current.incidenceSet$ do
        $Q.enqueue(< j, current >)$
    end for
end while
for each $E_F\ f \in M.keys$ do
    $\mathbb{Q}.add(CreateQuery(f, ""))$
end for
return $\mathbb{Q}$
```



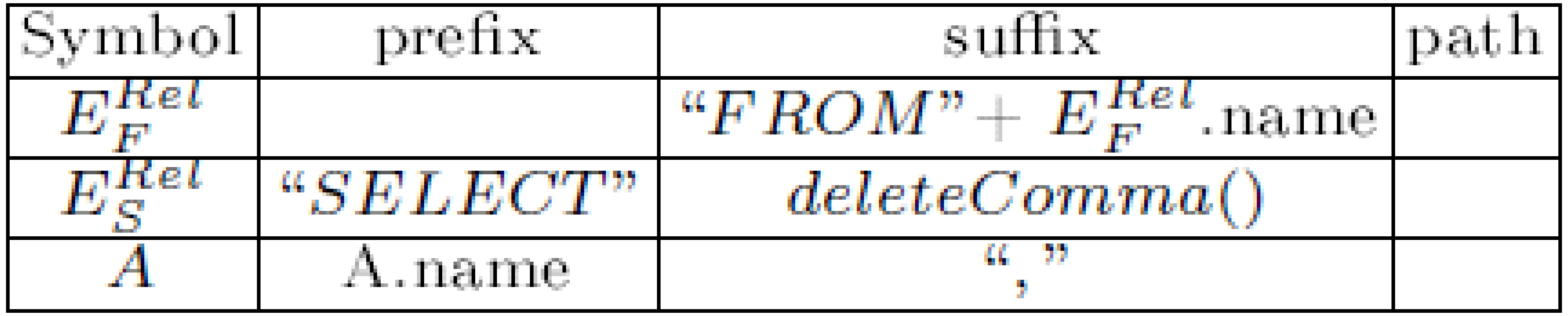
**Algorithm 2** Create Query algorithm

**Input:** $source\ E_H$, $path\ of\ E_H$ (adjacency list $M$ from Algorithm 1 is also available)
**Output:** A data store query $q$

```
$q \leftarrow prefixOf(source, path)$
for each $child \in M.get(source)$ do
    $q \leftarrow q + CreateQuery(child, pathOf(source))$
end for
$q \leftarrow q + suffixOf(source)$
return $q$
```

# Constraints & Rules - RDBMS

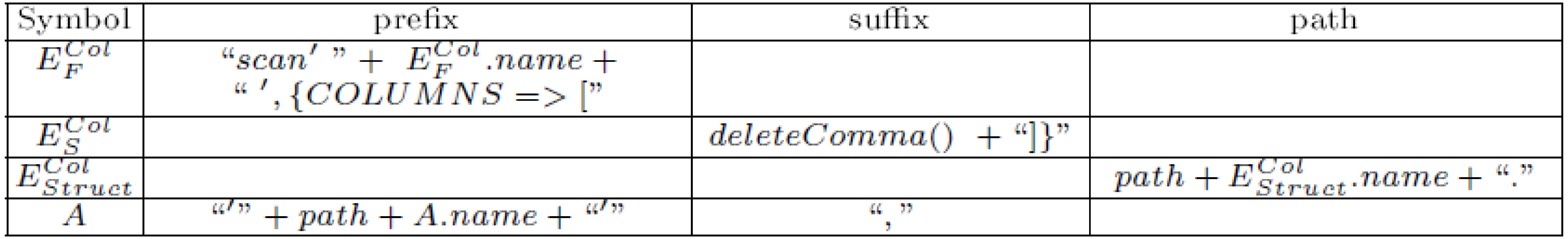
$$E_D^{Rel} \Rightarrow E_F^{Rel} *$$
$$E_F^{Rel} \Rightarrow E_S^{Rel}$$
$$E_S^{Rel} \Rightarrow A_C A *$$



| Symbol | prefix | suffix | path |
|---|---|---|---|
| $E_F^{Rel}$ |  | “$FROM$”+ $E_F^{Rel}$.name |  |
| $E_S^{Rel}$ | “$SELECT$” | $deleteComma()$ |  |
| $A$ | A.name | “,” |  |

# Constraints & Rules – Wide-Column Store

$$E_D^{Col} \Rightarrow E_F^{Col} *$$
$$E_F^{Col} \Rightarrow E_S^{Col}$$
$$E_S^{Col} \Rightarrow A_C E_{Struct}^{Col}$$
$$E_{Struct}^{Col} \Rightarrow A^+$$



| Symbol | prefix | suffix | path |
|---|---|---|---|
| $E_F^{Col}$ | “$scan'$ ” + $E_F^{Col}.name$ +<br>“ $'$, $\{COLUMNS => [$” | | |
| $E_S^{Col}$ | | $deleteComma()$ + “]}” | |
| $E_{Struct}^{Col}$ | | | $path + E_{Struct}^{Col}.name +$ “.” |
| $A$ | “$'$” $+ path + A.name +$ “$'$” | “, ” | |

# PostgreSQL Cost model background

- Measured on an arbitrary scale
- By default based on the cost of sequential page fetches
- Other variables are set with reference to that
- Can use and edit for a different scale
- "No well-defined method for determining ideal values"

# Cost model constants

- seq_page_cost (1.0): cost of disk page fetch that is part of a series of sequential fetches
- random_page_cost (4.0) : cost of non-sequentially-fetched disk page
  - raising it will make index scans more expensive
  - usually random access is more than 4 times expensive 4 is used under the assumption that majority of random access to disk like index reads are in cache
  - The default value could be thought of as random access is 40 times slower while expecting 90\% to be in the cache
- cpu_tuple_cost (0.01) : cost of processing each row during a query
- cpu_index_tuple_cost (0.005) : cost of processing each index entry
- cpu_operator_cost (0.0025) : cost of processing a each operator or function
- parallel_setup_cost (1000) cost of launching parallel worker process

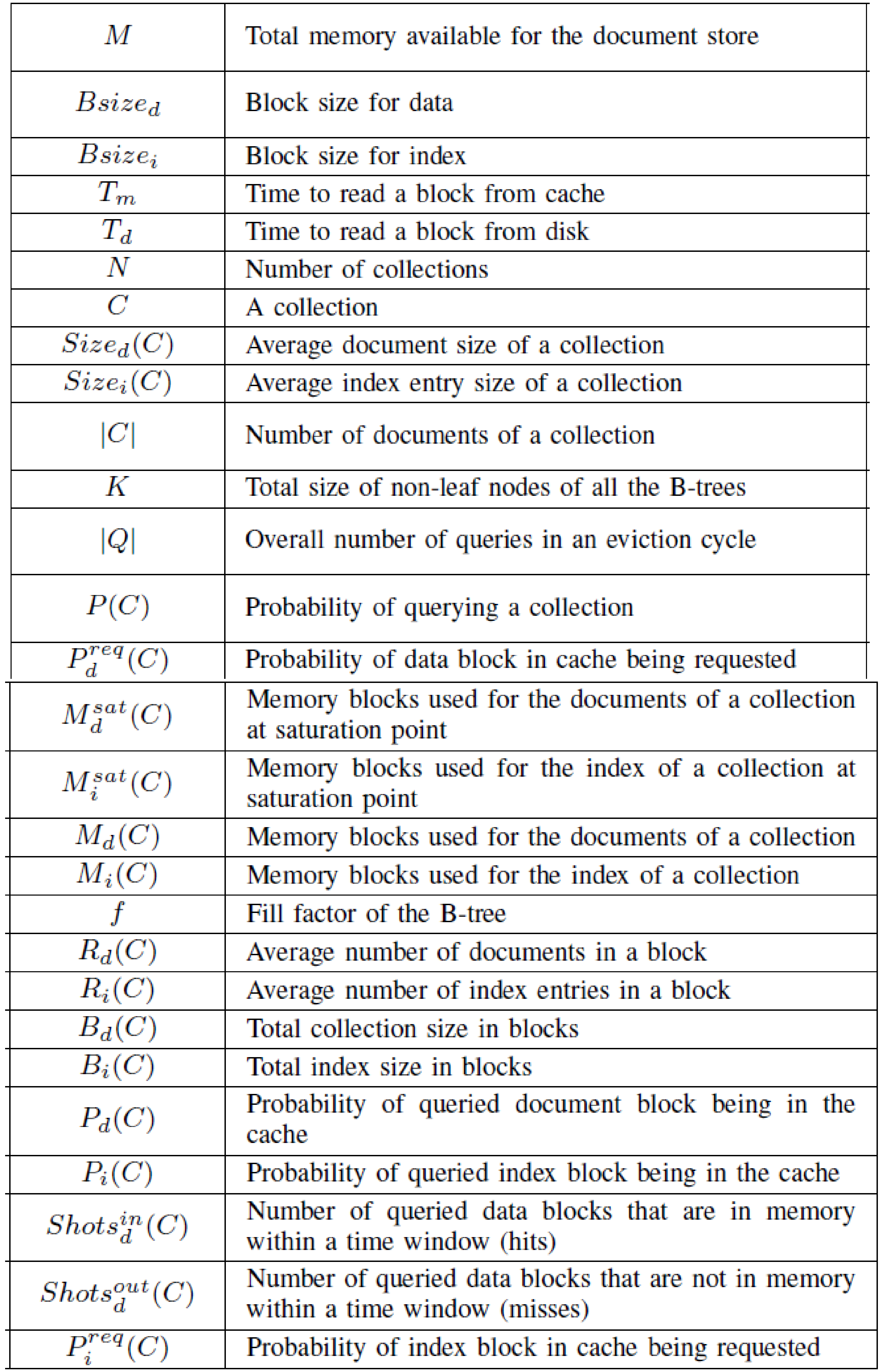
# Cost Model Constants

- parallel_tuple_cost (0.1) : cost of transferring one tuple from a parallel worker process to another process
- min_parallel_table_scan_size (8MB): minimum amount of table data that must be scanned for a parallel scan to be considered
- min_parallel_index_scan_size (512kB): minimum amount of index data that must be scanned in order for a parallel scan to be considered
- effective_cache_size (4GB) : effective size of the cache available for a single query

# Other cost models

- XML
    - Tree traversal from root
    - Page numbers from root to the value
- In memory database
    - Memory cost is difficult because its managed by the OS not the DBMS
    - Rely on number of tuples processed per operator

# Cost Model Contd. - MongoDB



| | |
|---|---|
| $M$ | Total memory available for the document store |
| $Bsize_d$ | Block size for data |
| $Bsize_i$ | Block size for index |
| $T_m$ | Time to read a block from cache |
| $T_d$ | Time to read a block from disk |
| $N$ | Number of collections |
| $C$ | A collection |
| $Size_d(C)$ | Average document size of a collection |
| $Size_i(C)$ | Average index entry size of a collection |
| $\|C\|$ | Number of documents of a collection |
| $K$ | Total size of non-leaf nodes of all the B-trees |
| $\|Q\|$ | Overall number of queries in an eviction cycle |
| $P(C)$ | Probability of querying a collection |
| $P_d^{req}(C)$ | Probability of data block in cache being requested |
| $M_d^{sat}(C)$ | Memory blocks used for the documents of a collection at saturation point |
| $M_i^{sat}(C)$ | Memory blocks used for the index of a collection at saturation point |
| $M_d(C)$ | Memory blocks used for the documents of a collection |
| $M_i(C)$ | Memory blocks used for the index of a collection |
| $f$ | Fill factor of the B-tree |
| $R_d(C)$ | Average number of documents in a block |
| $R_i(C)$ | Average number of index entries in a block |
| $B_d(C)$ | Total collection size in blocks |
| $B_i(C)$ | Total index size in blocks |
| $P_d(C)$ | Probability of queried document block being in the cache |
| $P_i(C)$ | Probability of queried index block being in the cache |
| $Shots_d^{in}(C)$ | Number of queried data blocks that are in memory within a time window (hits) |
| $Shots_d^{out}(C)$ | Number of queried data blocks that are not in memory within a time window (misses) |
| $P_i^{req}(C)$ | Probability of index block in cache being requested |

$$Req(C) = |Q| * P(C)$$

$$E(C) = |C| * \left(1 - \left(\frac{|C| - 1}{|C|}\right)^{Req(C)}\right)$$

$$P_d^{req}(C) = 1 - (1 - SF(C))^{R_d(C)}$$

$$P_i^{req}(C) = 1 - (1 - SF(C))^{R_i(C)}$$

$$M_d^{sat}(C) = \left\lceil \frac{|C|}{R_d(C)} \right\rceil * P_d^{req}(C)$$

$$M_i^{sat}(C) = \left\lceil \frac{|C|}{R_i(C)} \right\rceil * P_i^{req}(C)$$

$$M = \sum_{j=1}^{N} (M_d^{sat}(C_j) * Bsize_d + M_i^{sat}(C_j) * Bsize_i) + K$$

$$Shots_{in}(C) = P(C) \cdot |Q| \cdot \frac{M_d(C)}{B_d(C)}$$

$$E_d(C) = M_d(C)\left(1 - \frac{1}{M_d(C)}\right)^{Shots_{in}(C)} \approx M_d(C) \cdot e^{-\frac{P(C) \cdot |Q|}{B_d(C)}}$$

$$P_d^{out}(C) = \frac{W_d(C) \cdot E_d(C)}{\sum_{j=1}^{N} (W_d(C_j) \cdot E_d(C_j) + W_d(C_j) \cdot E_i(C_j))}$$

$$W_d(C) = \frac{picks_d(C)}{queue\ size}$$

$$picks_d(C) = queue\ size \cdot \frac{M_d(C) \cdot Bsize_d}{M}$$

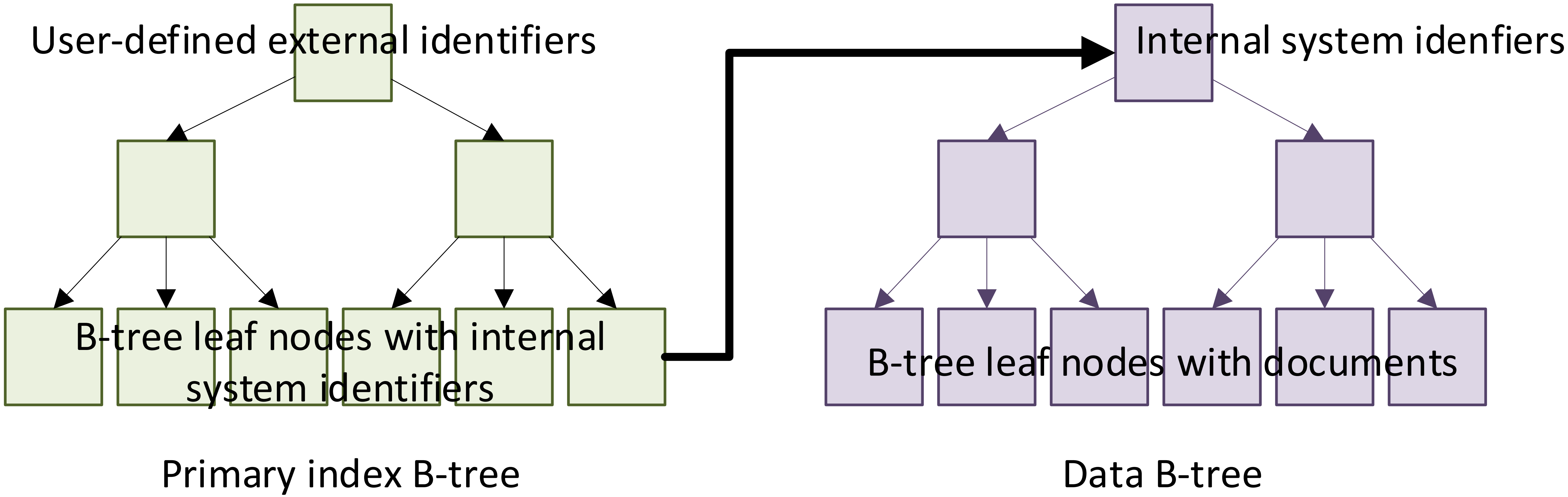
$$\therefore W_d(C) = \frac{M_d(C) \cdot Bsize_d}{M}$$

$$P_d^{in}(C) = \frac{P(C) \cdot (1 - P_d(C))}{\sum_{j=1}^{N} (P(C_j) \cdot (1 - P_d(C_j)) + P(C_j) \cdot (1 - P_i(C_j)))}$$

$$\forall C_j : P_d^{in}(C_j) = P_d^{out}(C_j), \qquad P_i^{in}(C_j) = P_i^{out}(C_j)$$

$$M = \sum_{j=1}^{N} (M_d(C_j) * Bsize_d + M_i(C_j) * Bsize_i) + K$$

# MongoDB B-trees



User-defined external identifiers

B-tree leaf nodes with internal system identifiers

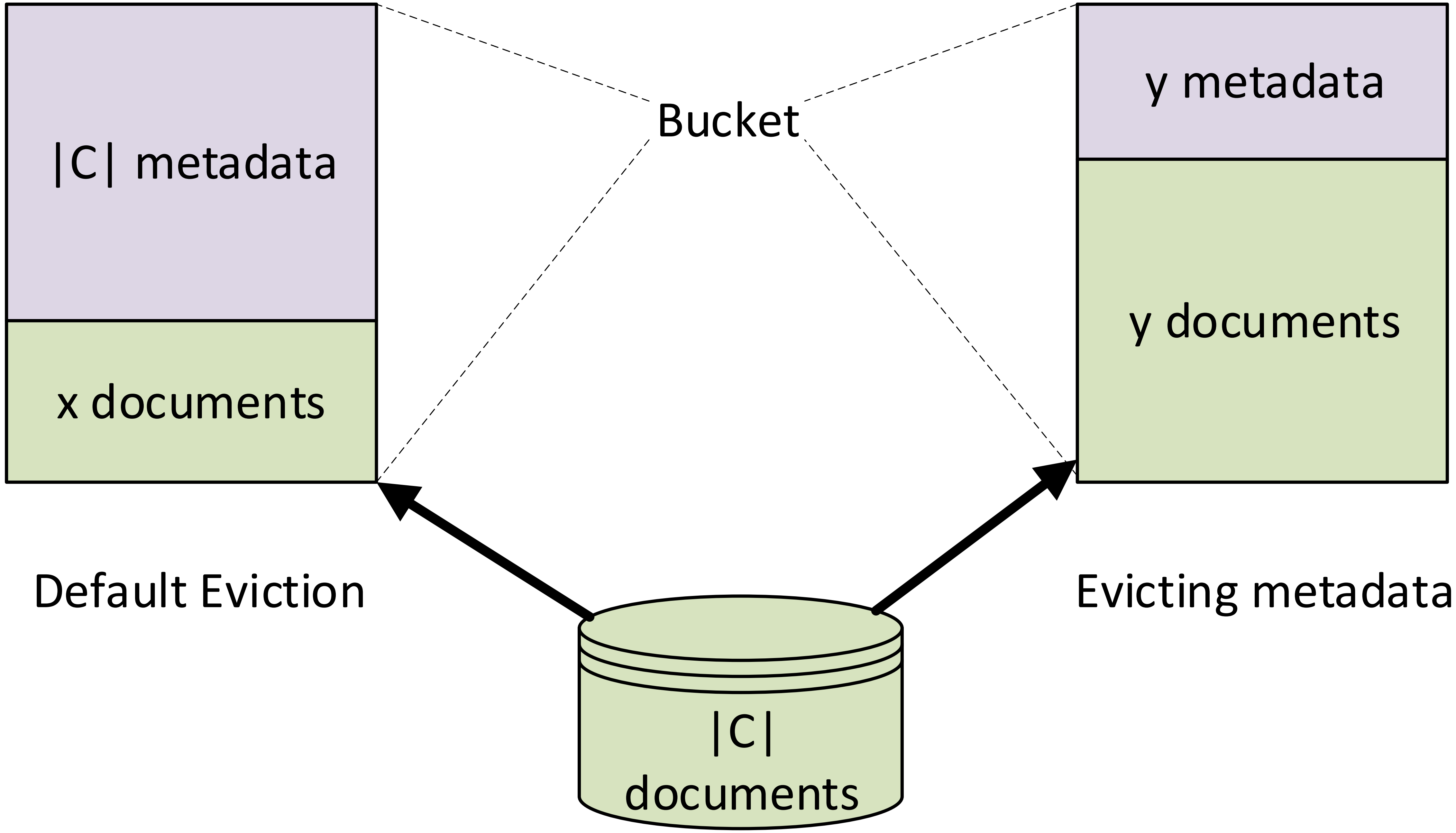
Primary index B-tree

Internal system idenfiers

B-tree leaf nodes with documents

Data B-tree

# Couchbase Memory


|C| metadata
x documents
Bucket
y metadata
y documents
Default Eviction
Evicting metadata
|C|
documents