Compressed Indexes for Fast Search of Semantic Data

Raffaele Perego
ISTI-CNR
Pisa, Italy

Giulio Ermanno Pibiri
ISTI-CNR
Pisa, Italy

Rossano Venturini
The University of Pisa
Pisa, Italy

ISTITUTO DI SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE "A. FAEDO"

IN SUPREMÆ DIGNITATIS
1343

The 10-th Italian Information Retrieval Workshop (IIR 2019)

17/09/2019

Resource Description Framework (RDF)

“RDF is a standard model for data interchange on the Web.”
Source: https://www.w3.org/RDF

Statements are encoded with **triples**:
Subject (**S**) - Predicate (**P**) - Object (**O**)

Resource Description Framework (RDF)

“RDF is a standard model for data interchange on the Web.”
Source: https://www.w3.org/RDF

Statements are encoded with **triples**:
Subject (**S**) - Predicate (**P**) - Object (**O**)

“Bob Smith knows John Doe.”

↓

```
<http://example.name#BobSmith12> <http://xmlns.com/foaf/0.1/knows> <http://example.name#JohnDoe34>
```

The problem

Huge datasets: **billions** of triples.

Storage space is an issue:
compression is mandatory.

How to support triple selection patterns (with wildcards) **efficiently**?

The problem

Huge datasets: **billions** of triples.

Storage space is an issue:
compression is mandatory.

How to support triple selection patterns (with wildcards) **efficiently**?

<Bob Smith> <knows> <???>

<???> <???> John Doe

<Bob Smith> <???> <Sara Parker>

The problem

Huge datasets: **billions** of triples.

Storage space is an issue:
compression is mandatory.

How to support triple selection patterns (with wildcards) **efficiently**?

<Bob Smith> <knows> <???>

<???> <???> John Doe

<Bob Smith> <???> <Sara Parker>

1 wildcard:
SP?
S?O
?PO

2 wildcards:
S??
?P?
??O

3 wildcards:
???

0 wildcard:
SPO

State-of-the-art solutions

Too costly in terms of **space**.

- Materialize **all** possible S-P-O permutations (6 separate indexes).
- Do **not** use sophisticated compression techniques.
- Expensive additional indexes to support retrieval.

The Permuted Trie Index: preliminaries

Map URI strings to integers to reduce space requirements:
we deal with datasets of integer triples.

Selection patterns

S P O
S P ?
S ? ?
? ? ?

? P O
? P ?

S ? O
? ? O

The Permuted Trie Index: preliminaries

Map URI strings to integers to reduce space requirements:
we deal with datasets of integer triples.

Selection patterns

S P O
S P ?
S ? ?
? ? ?

→ **S-P-O** order

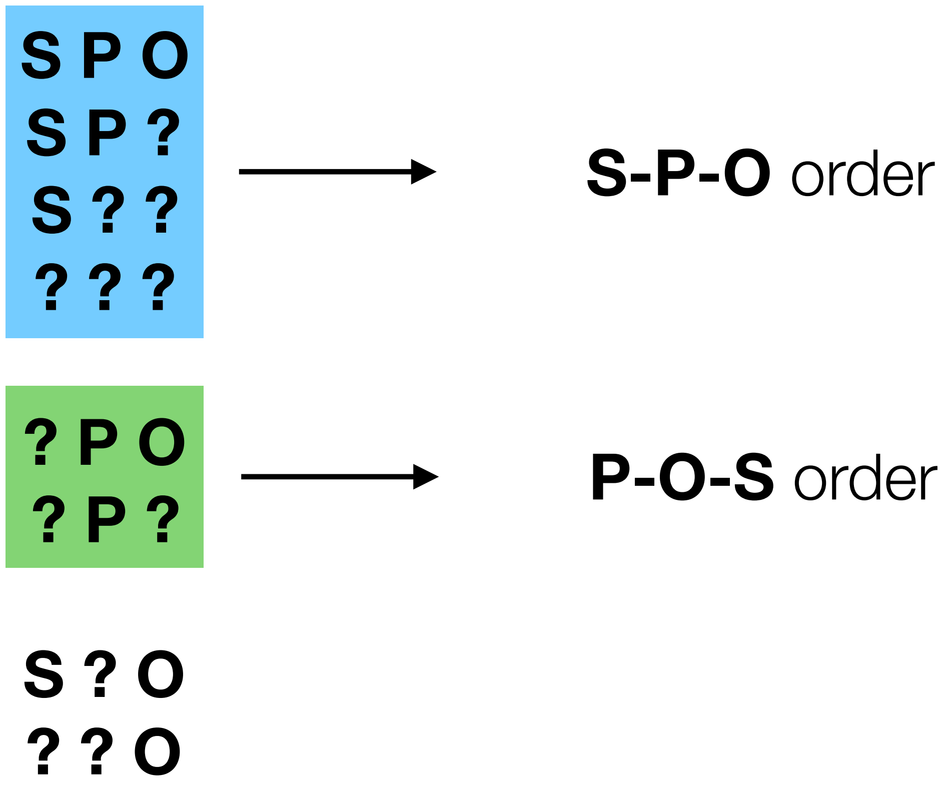
? P O
? P ?

S ? O
? ? O

The Permuted Trie Index: preliminaries

Map URI strings to integers to reduce space requirements:
we deal with datasets of integer triples.

Selection patterns



S P O
S P ?
S ? ?
? ? ?

→ **S-P-O** order

? P O
? P ?

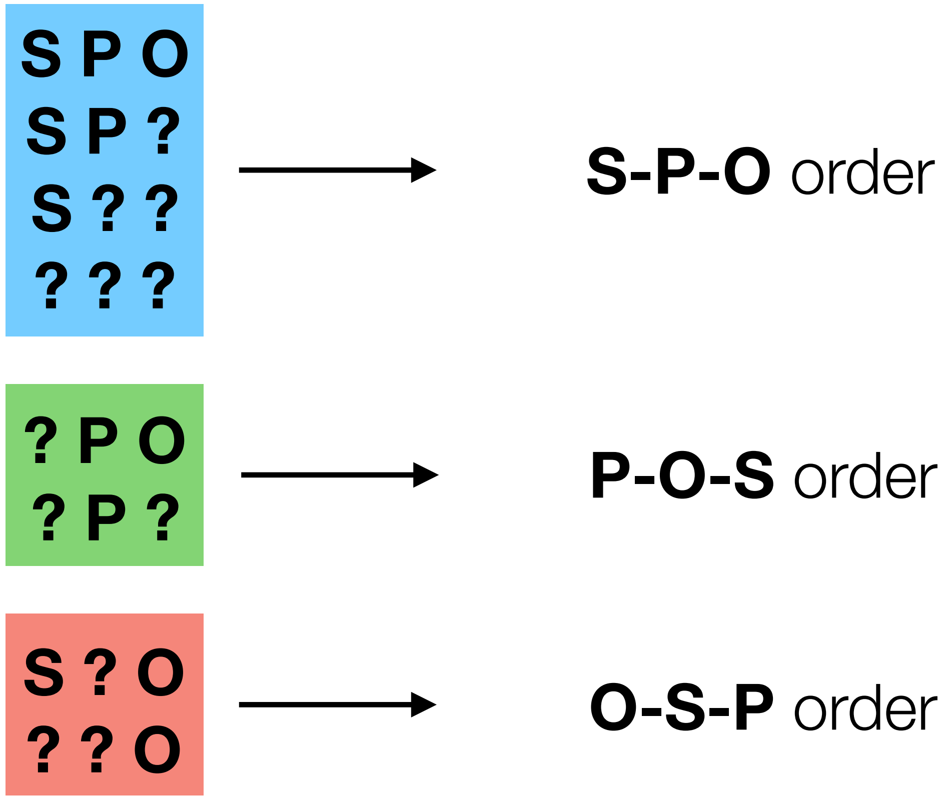
→ **P-O-S** order

S ? O
? ? O

The Permuted Trie Index: preliminaries

Map URI strings to integers to reduce space requirements:
we deal with datasets of integer triples.

Selection patterns



S P O
S P ?
S ? ?
? ? ?

S-P-O order

? P O
? P ?

P-O-S order

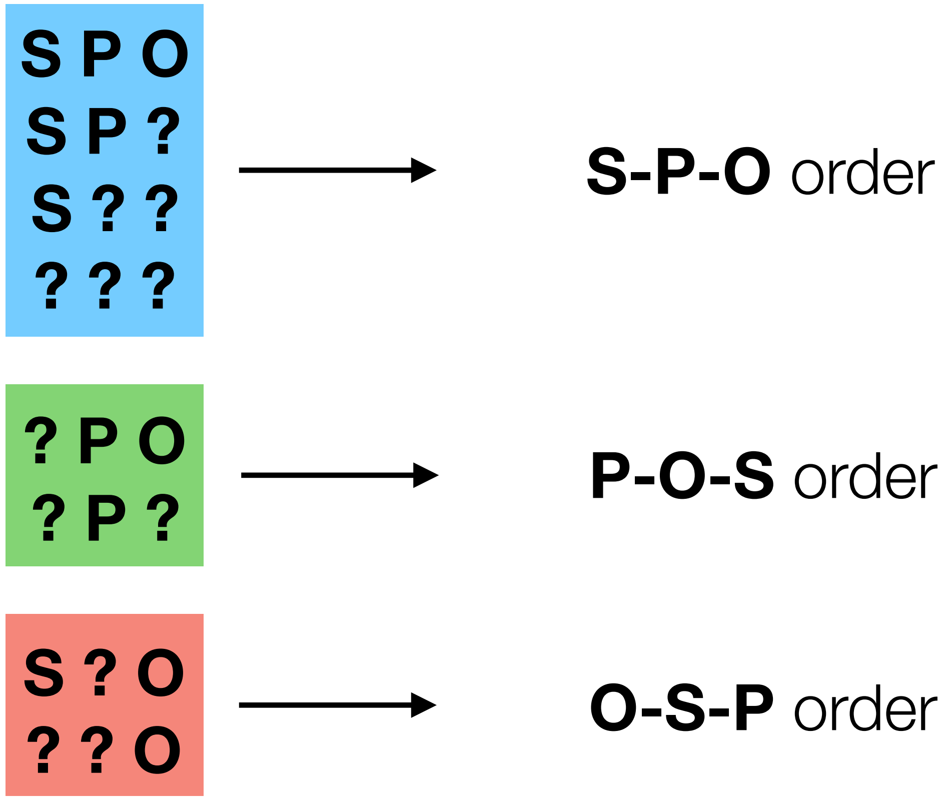
S ? O
? ? O

O-S-P order

The Permuted Trie Index: preliminaries

Map URI strings to integers to reduce space requirements:
we deal with datasets of integer triples.

Selection patterns



S P O
S P ?
S ? ?
? ? ?

S-P-O order

? P O
? P ?

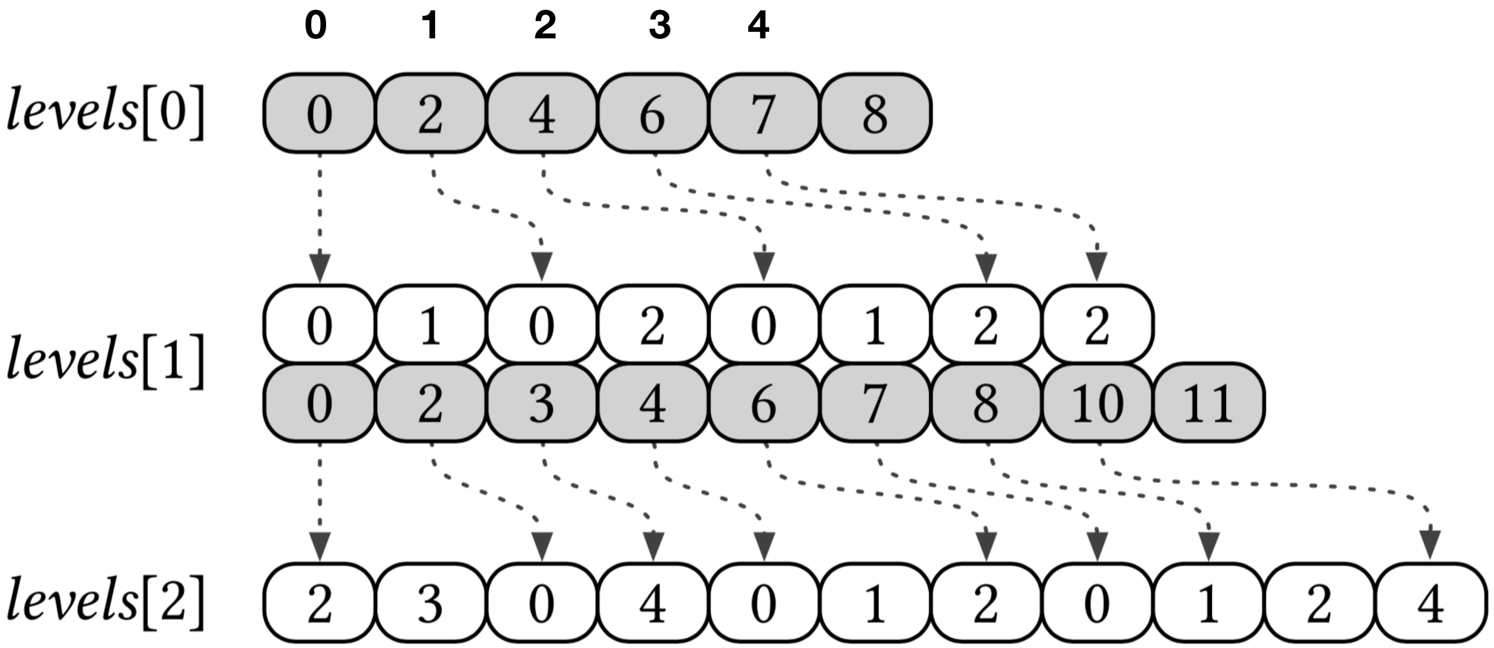
P-O-S order

S ? O
? ? O

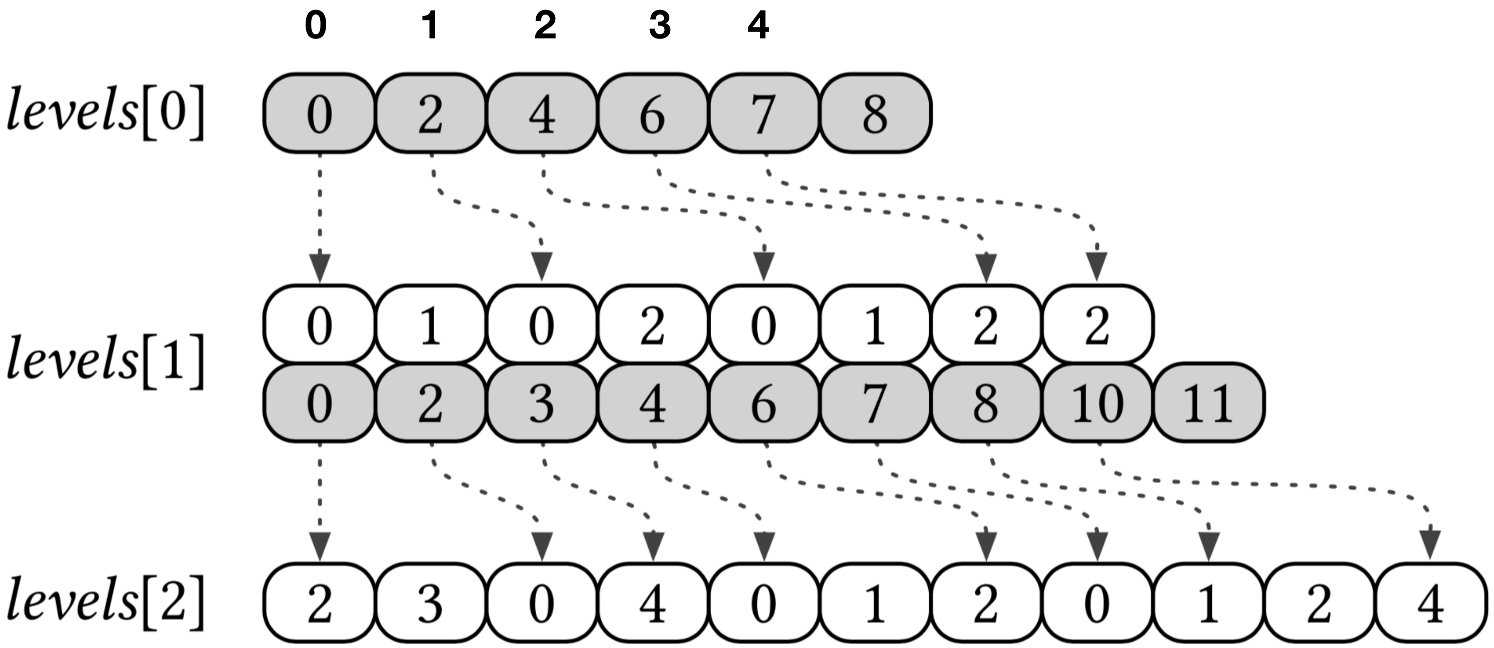
O-S-P order

Store an **integer trie**
data structure
for each permutation.

The Permuted Trie Index: organisation

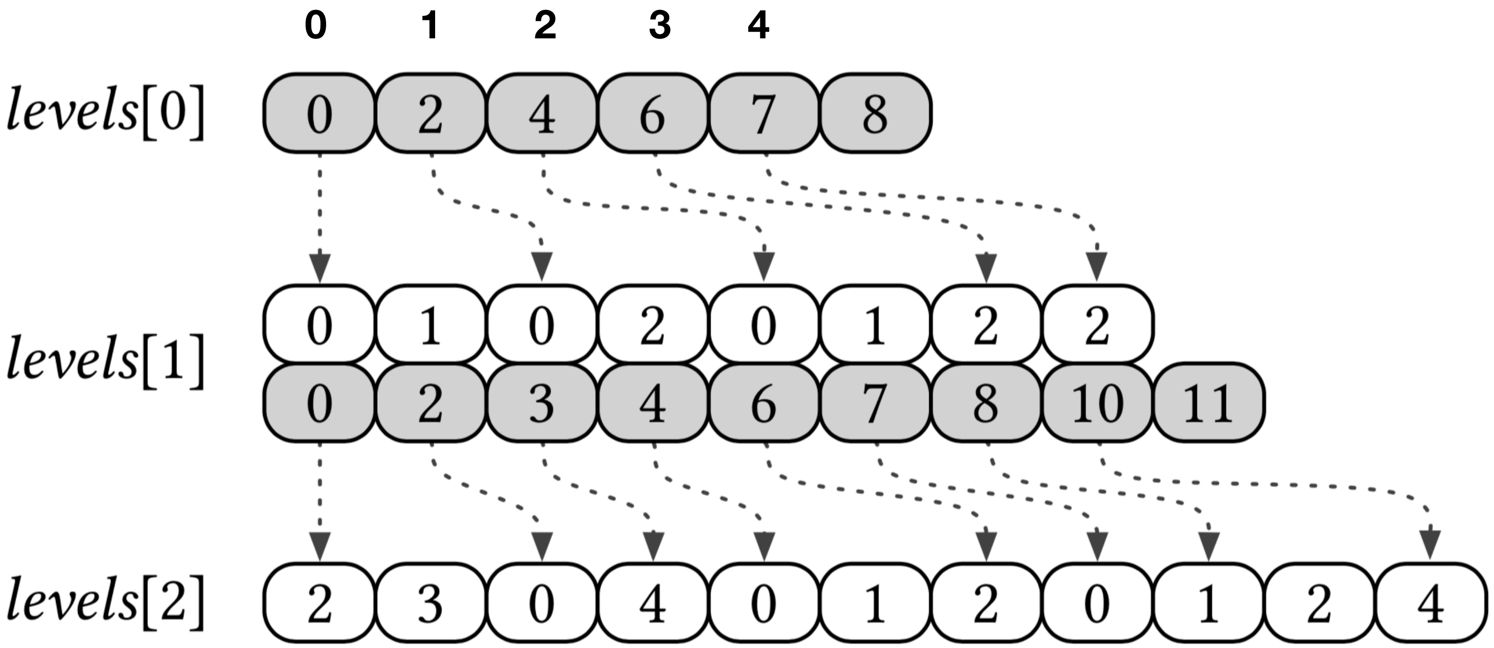


The Permuted Trie Index: organisation



- **Common prefixes** are encoded once.
- Two integer **sequences** per level (nodes and pointers).
- Symmetrically support **all** selection patterns with 1 and 2 wildcards.
- **Cache-friendly** memory layout.

The Permuted Trie Index: organisation

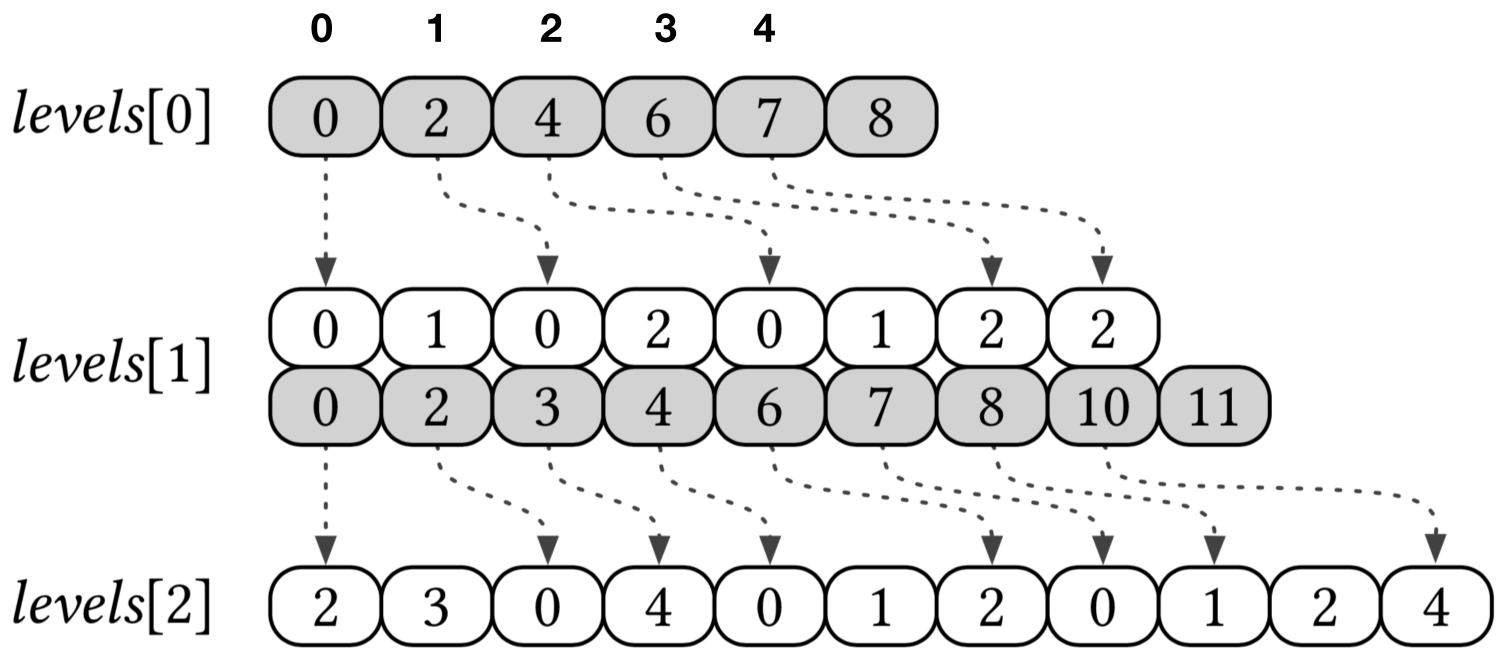


- **Common prefixes** are encoded once.
- Two integer **sequences** per level (nodes and pointers).

Allows effective compression

- Symmetrically support **all** selection patterns with 1 and 2 wildcards.
- **Cache-friendly** memory layout.

The Permuted Trie Index: organisation



- **Common prefixes** are encoded once.
- Two integer **sequences** per level (nodes and pointers).

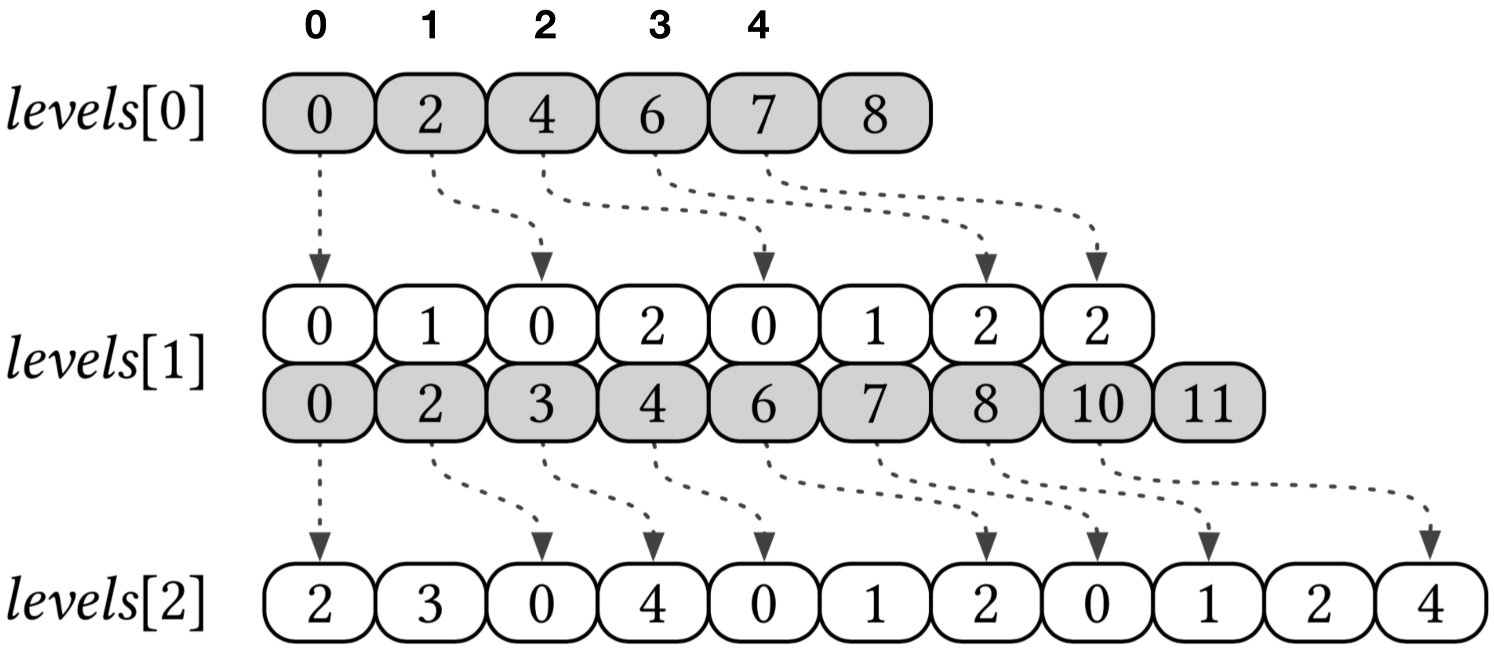
Allows effective compression

- Symmetrically support **all** selection patterns with 1 and 2 wildcards.
- **Cache-friendly** memory layout.

Fast retrieval

The Permuted Trie Index: organisation

(1, 2, ?)



- **Common prefixes** are encoded once.
- Two integer **sequences** per level (nodes and pointers).

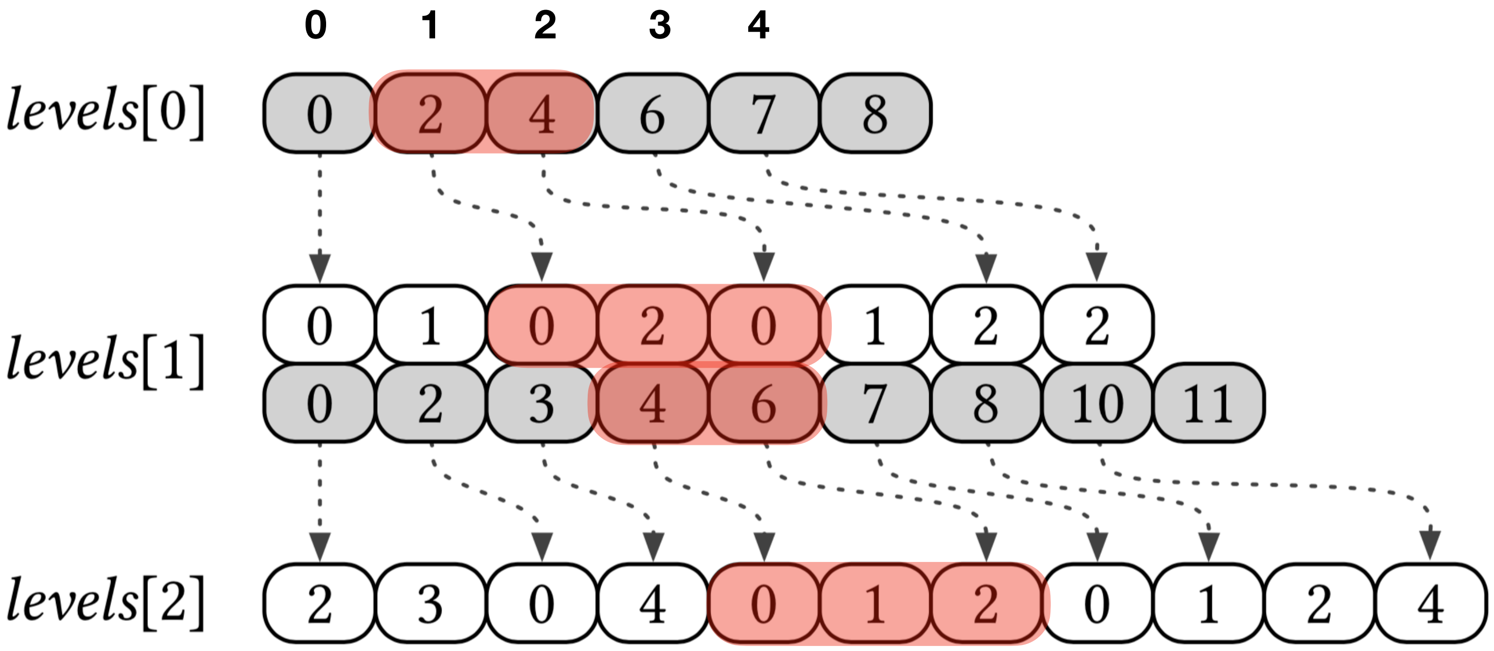
Allows effective compression

- Symmetrically support **all** selection patterns with 1 and 2 wildcards.
- **Cache-friendly** memory layout.

Fast retrieval

The Permuted Trie Index: organisation

(1, 2, ?)



- **Common prefixes** are encoded once.
- Two integer **sequences** per level (nodes and pointers).

Allows effective compression

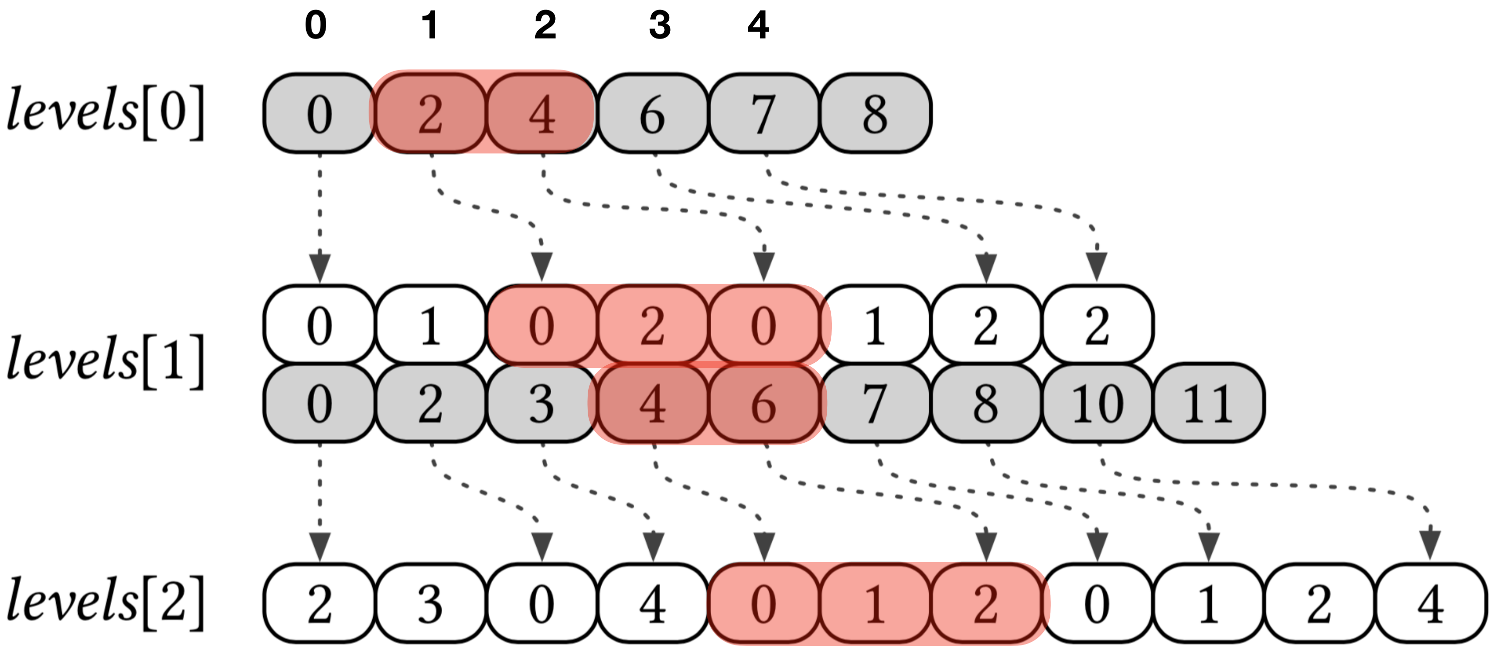
- Symmetrically support **all** selection patterns with 1 and 2 wildcards.
- **Cache-friendly** memory layout.

Fast retrieval

The Permuted Trie Index: organisation

(1, 2, ?)

(1, 2, 0)
(1, 2, 1)



- **Common prefixes** are encoded once.
- Two integer **sequences** per level (nodes and pointers).

Allows effective compression

- Symmetrically support **all** selection patterns with 1 and 2 wildcards.
- **Cache-friendly** memory layout.

Fast retrieval

The Permuted Trie Index: refinements

1. Cross Compression

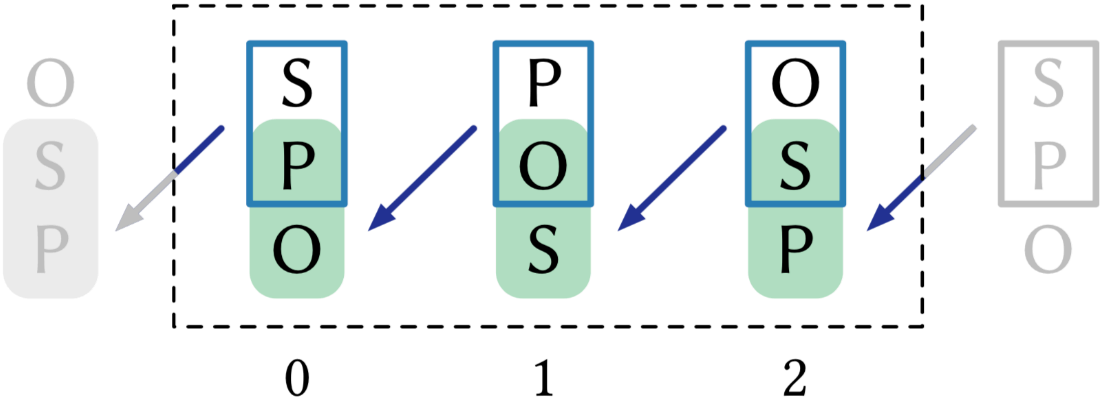
2. Permutation Elimination

Cross Compression

Fact: the **same** triple appears three times, but in **different** permutations.

Cross Compression

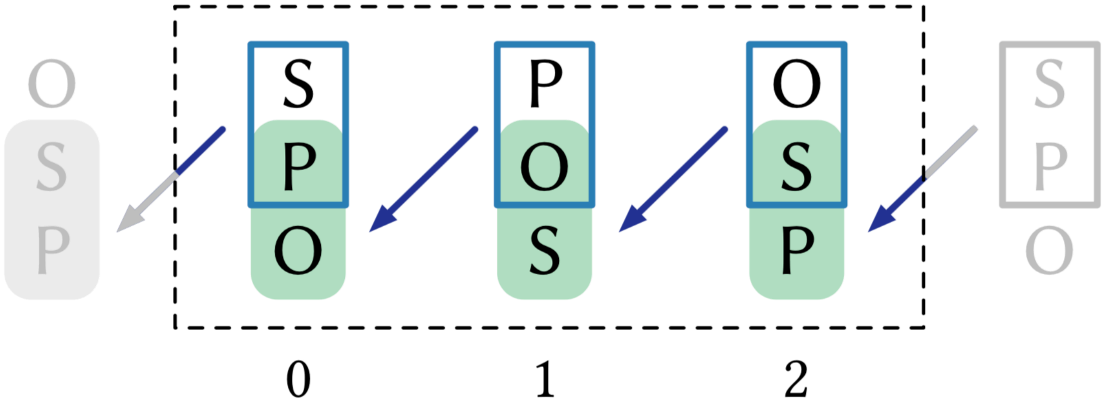
Fact: the **same** triple appears three times, but in **different** permutations.



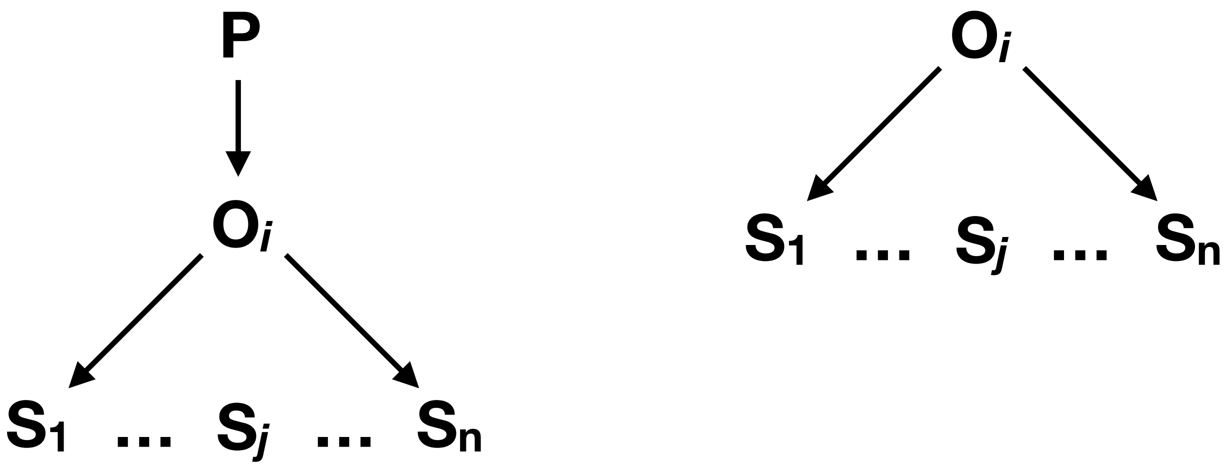
We can represent the subjects in trie **1** by using the subjects in trie **2**.

Cross Compression

Fact: the **same** triple appears three times, but in **different** permutations.

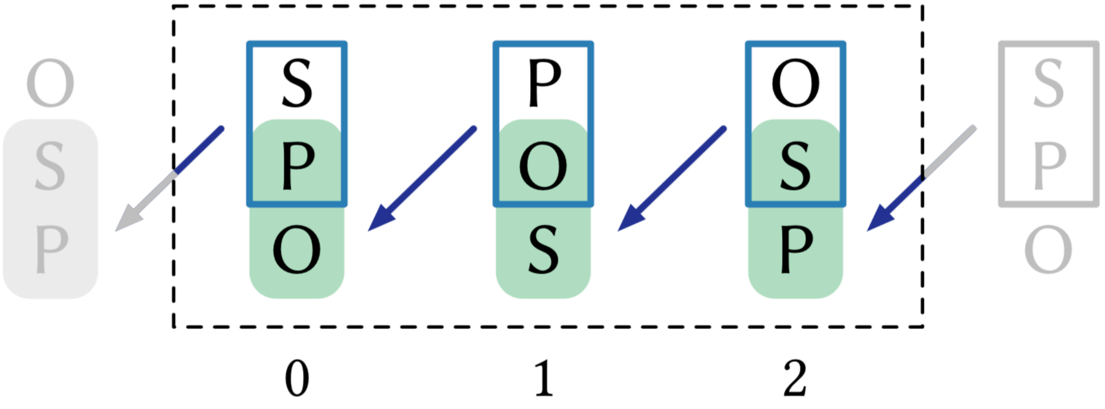


We can represent the subjects in trie **1** by using the subjects in trie **2**.



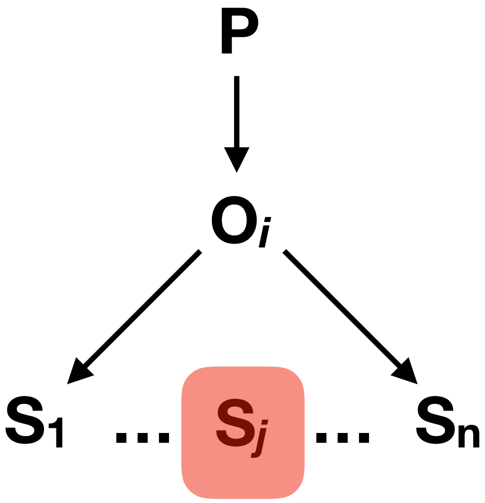
Cross Compression

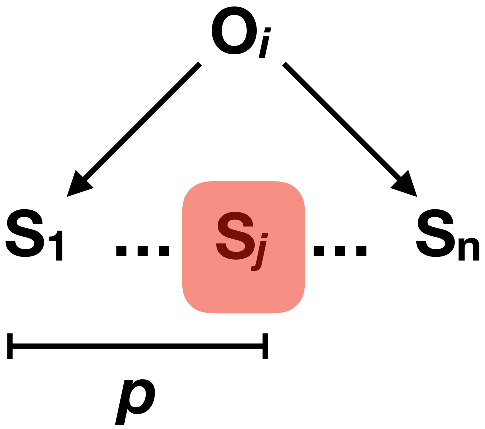
Fact: the **same** triple appears three times, but in **different** permutations.



We can represent the subjects in trie **1** by using the subjects in trie **2**.

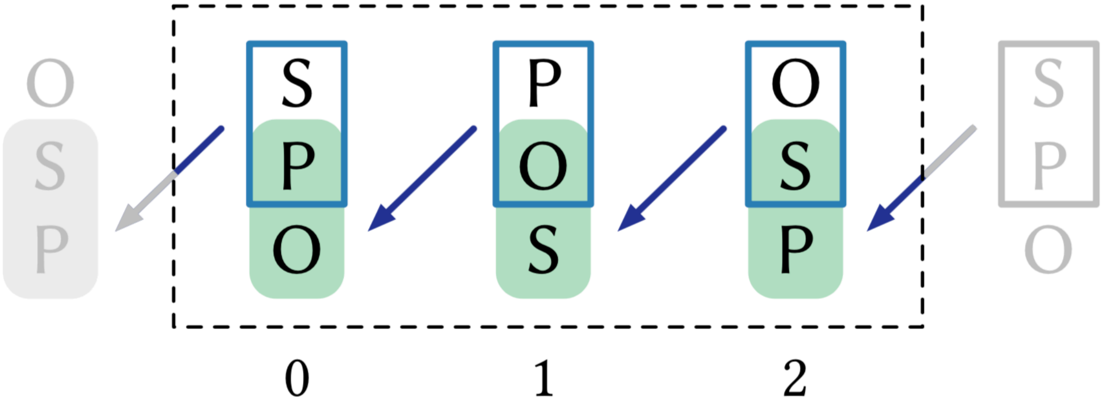
Represent **S_j** as its position ***p***.





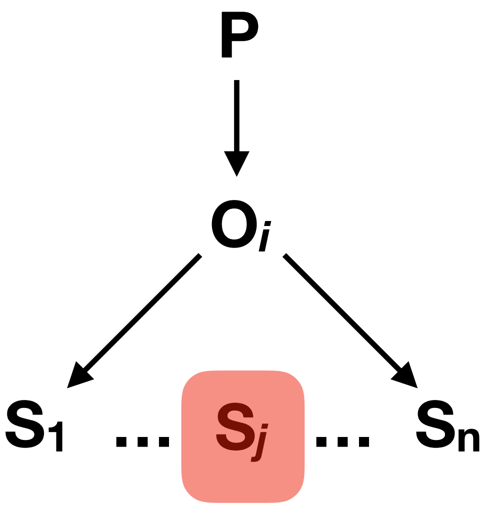
Cross Compression

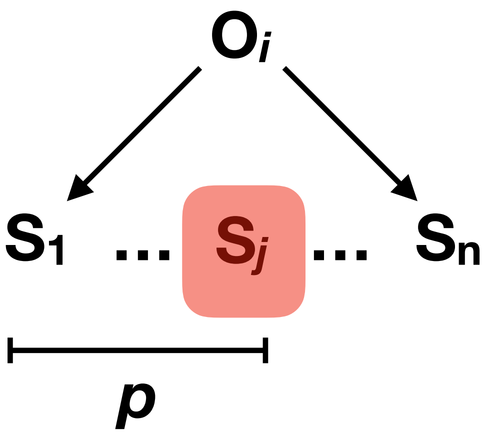
Fact: the **same** triple appears three times, but in **different** permutations.



We can represent the subjects in trie **1** by using the subjects in trie **2**.

Represent $\mathbf{S}_j$ as its position $\boldsymbol{p}$.





Why?

Trie	Level	Average	Maximum
SPO	1	5.54	52
	2	2.32	8489
POS	1	91,578.32	21,219,244
	2	2.59	10,141,311
OSP	1	2.70	10,141,327
	2	1.13	10

Number of children in Dbpedia.

Permutation Elimination

Fact: predicates are **few**, thus **S?O** returns only few matches.

Permutation Elimination

Fact: predicates are **few**, thus **S?O** returns only few matches.

We can **pattern match S?O on the SPO trie**,
instead of the OSP trie.

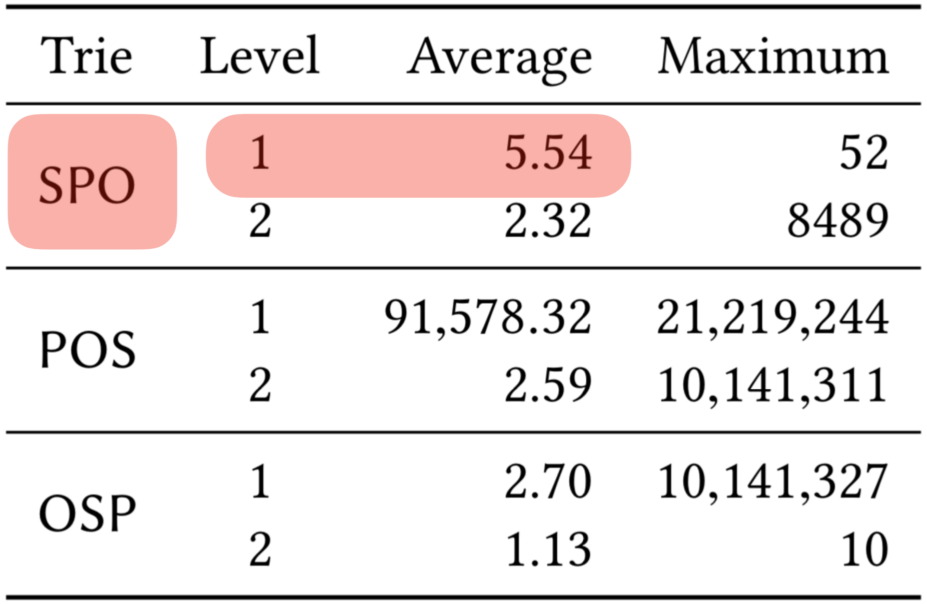
Given a (s,o) pair: for each child p_i of s,
check is o is a child of p_i. If so, then (s,p_i,o) is a match.

Permutation Elimination

Fact: predicates are **few**, thus **S?O** returns only few matches.

We can **pattern match S?O on the SPO trie**,
instead of the OSP trie.

Given a (s,o) pair: for each child p_i of s,
check is o is a child of p_i. If so, then (s,p_i,o) is a match.



Trie	Level	Average	Maximum
SPO	1	5.54	52
	2	2.32	8489
POS	1	91,578.32	21,219,244
	2	2.59	10,141,311
OSP	1	2.70	10,141,327
	2	1.13	10

Number of children in Dbpedia.

Less than 6 checks are needed on average!

Permutation Elimination

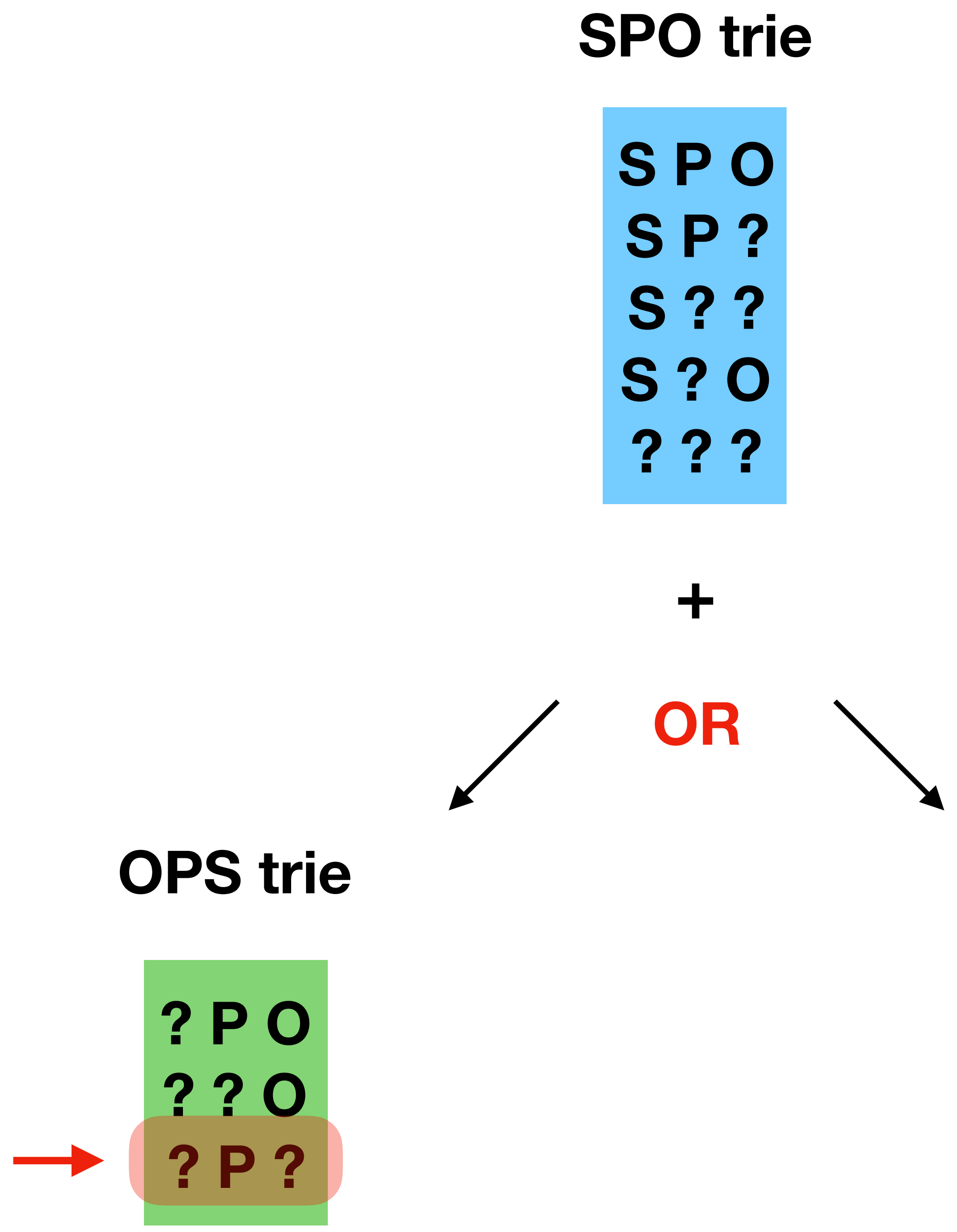
SPO trie

S P O
S P ?
S ? ?
S ? O
? ? ?

+

OR

Permutation Elimination



SPO trie

S P O
S P ?
S ? ?
S ? O
? ? ?

+

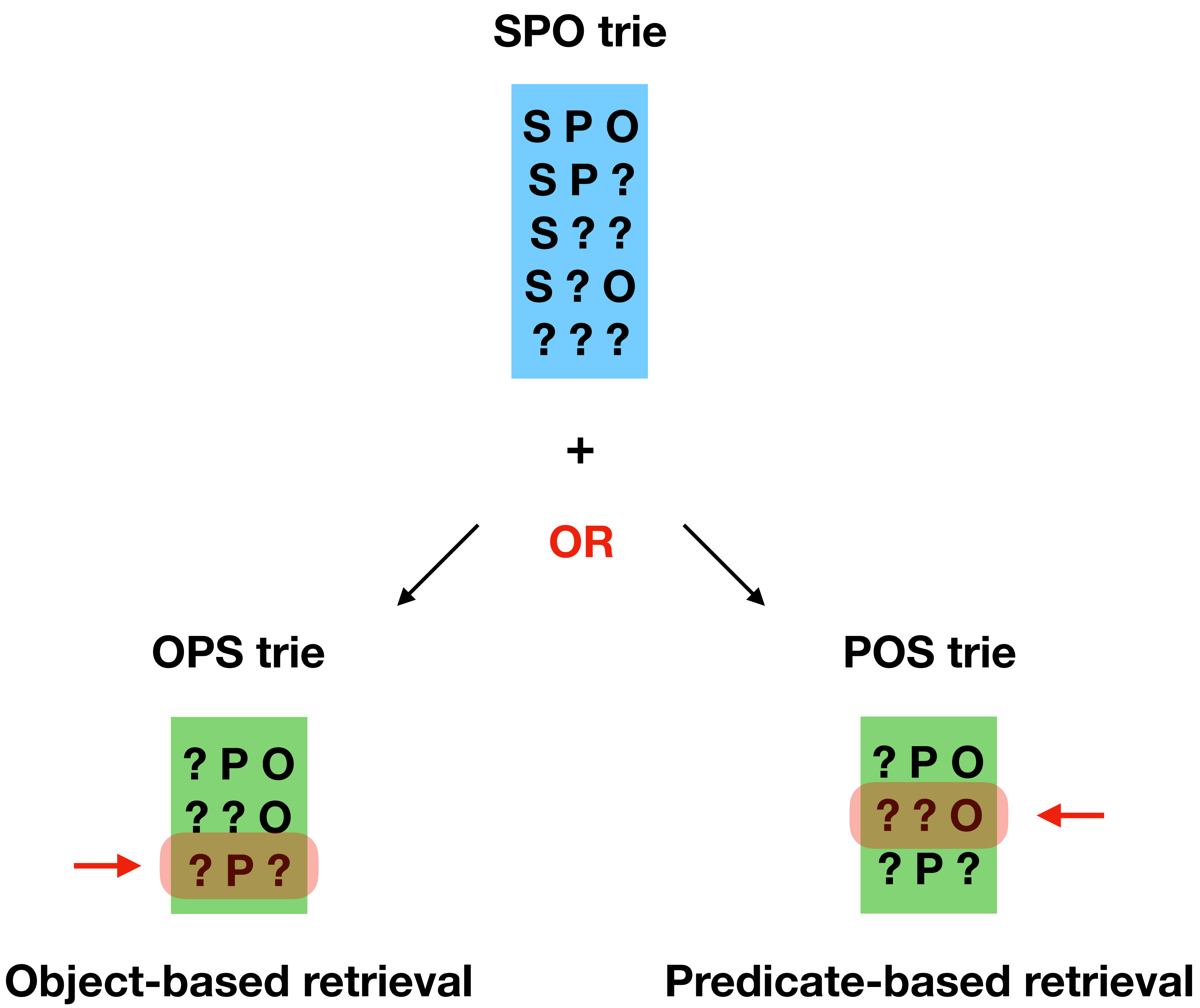
OR

OPS trie

? P O
? ? O
? P ?

Object-based retrieval

Permutation Elimination



SPO trie

S P O
S P ?
S ? ?
S ? O
? ? ?

+

OR

OPS trie

? P O
? ? O
? P ?

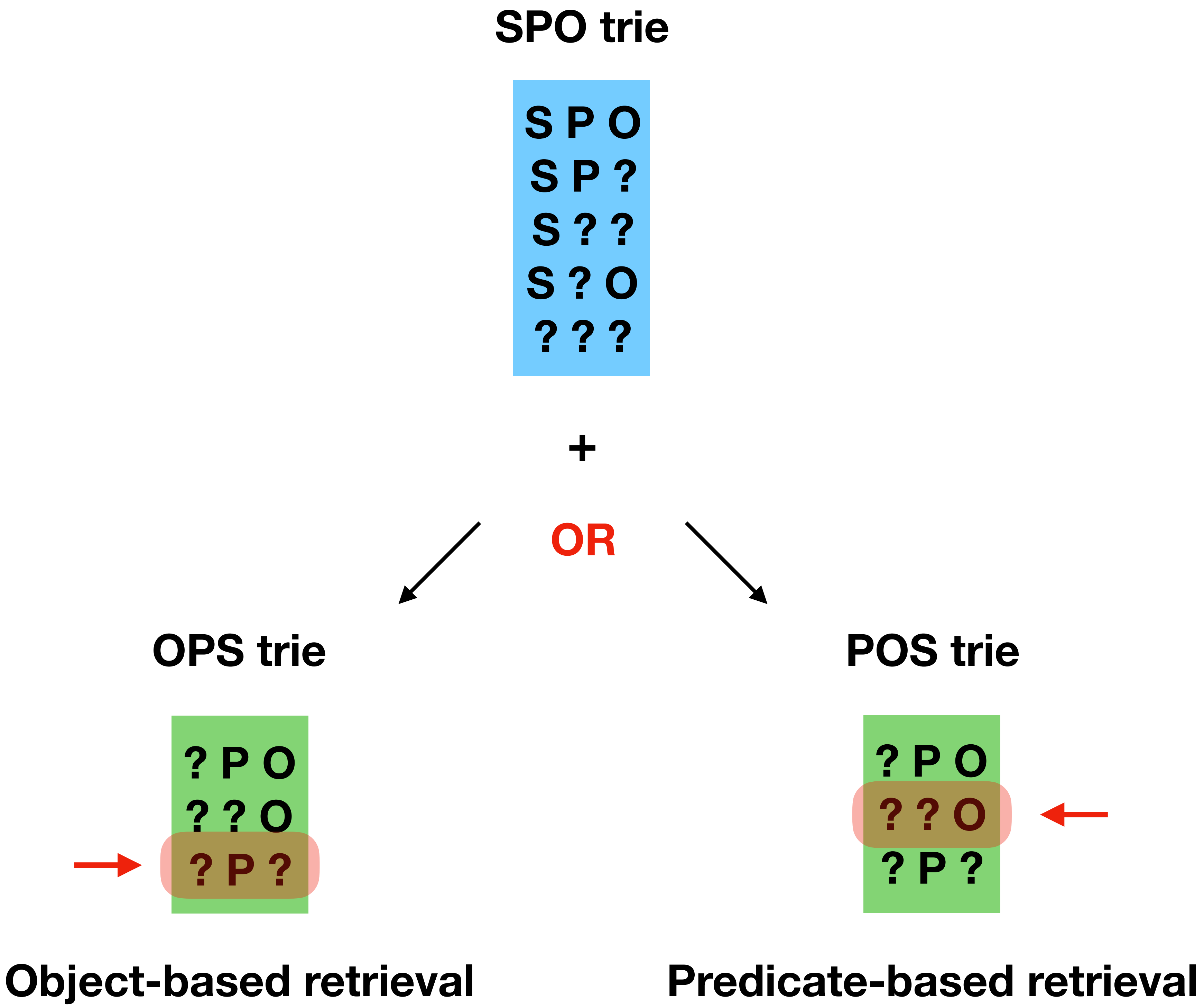
POS trie

? P O
? ? O
? P ?

Object-based retrieval

Predicate-based retrieval

Permutation Elimination



SPO trie

S P O
S P ?
S ? ?
S ? O
? ? ?

+

OR

OPS trie

? P O
? ? O
? P ?

POS trie

? P O
? ? O
? P ?

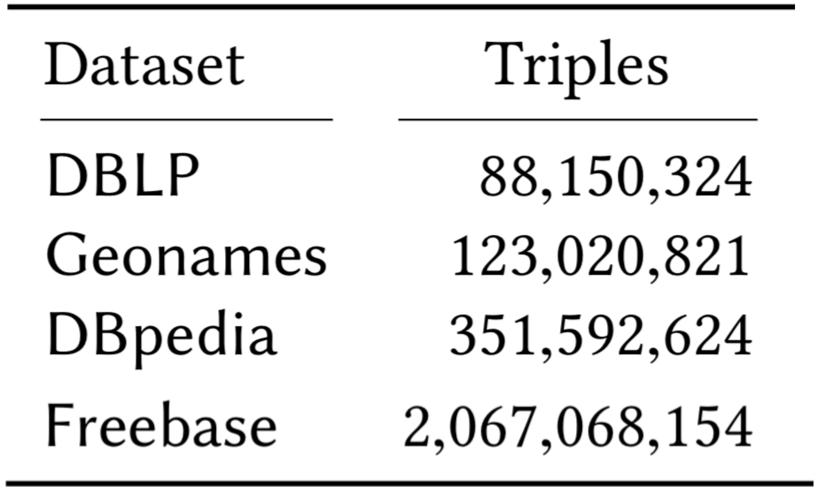
Object-based retrieval

Predicate-based retrieval

We can eliminate a permutation, thus saving 1/3 of the space of the index.

Experiments: setting

Datasets



Dataset	Triples
DBLP	88,150,324
Geonames	123,020,821
DBpedia	351,592,624
Freebase	2,067,068,154

Machine

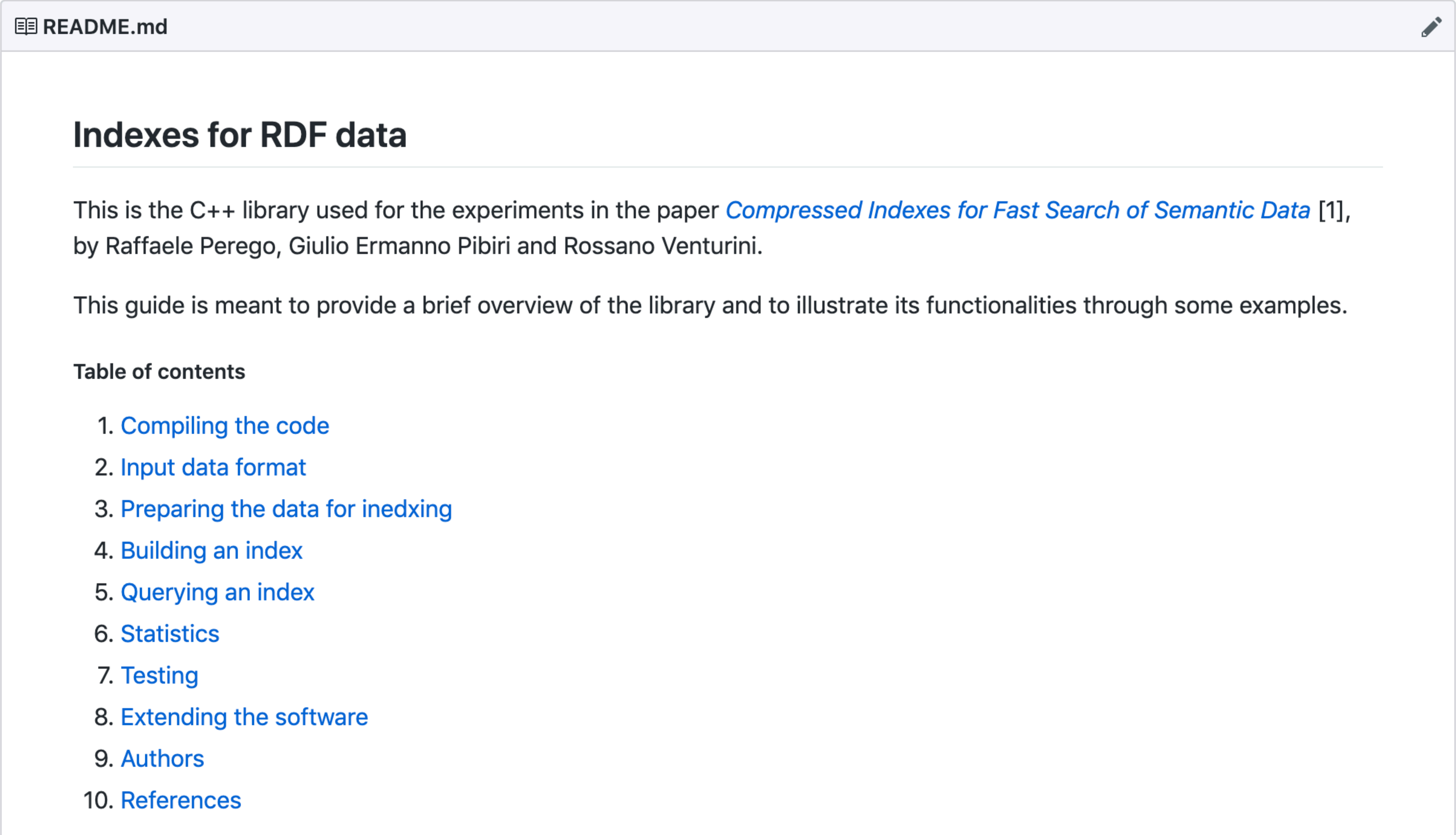
i7-7700 CPU (@3.6 GHz), 64 GB of RAM DDR3 (@2.133 GHz)
Linux 4.4.0, 64 bits

Compiler

`gcc` 7.2.0 (with all optimizations)

Experiments: C++ code

C++ code at https://github.com/jermp/rdf_indexes



README.md

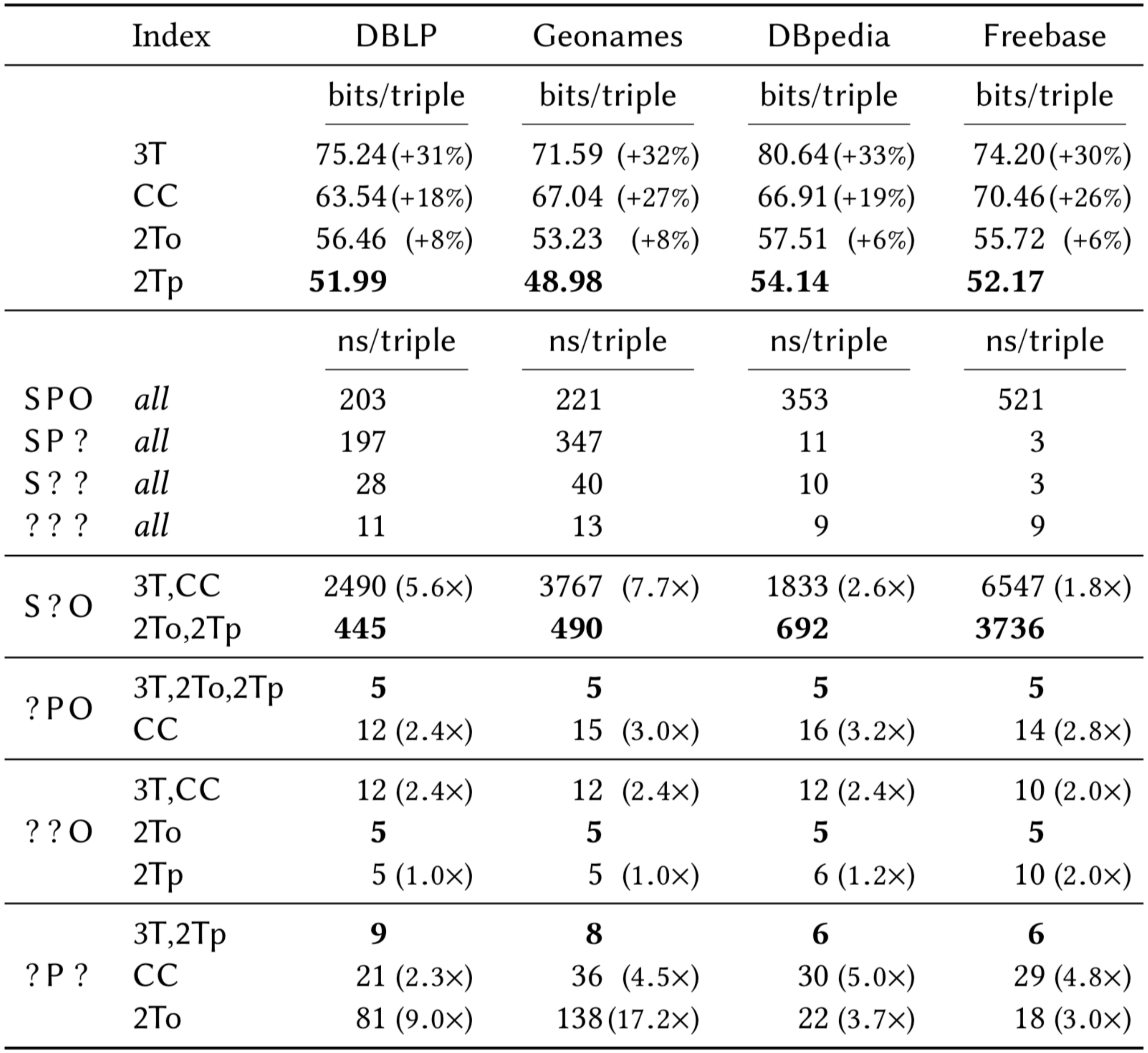
Indexes for RDF data

This is the C++ library used for the experiments in the paper *Compressed Indexes for Fast Search of Semantic Data* [1], by Raffaele Perego, Giulio Ermanno Pibiri and Rossano Venturini.

This guide is meant to provide a brief overview of the library and to illustrate its functionalities through some examples.

Table of contents

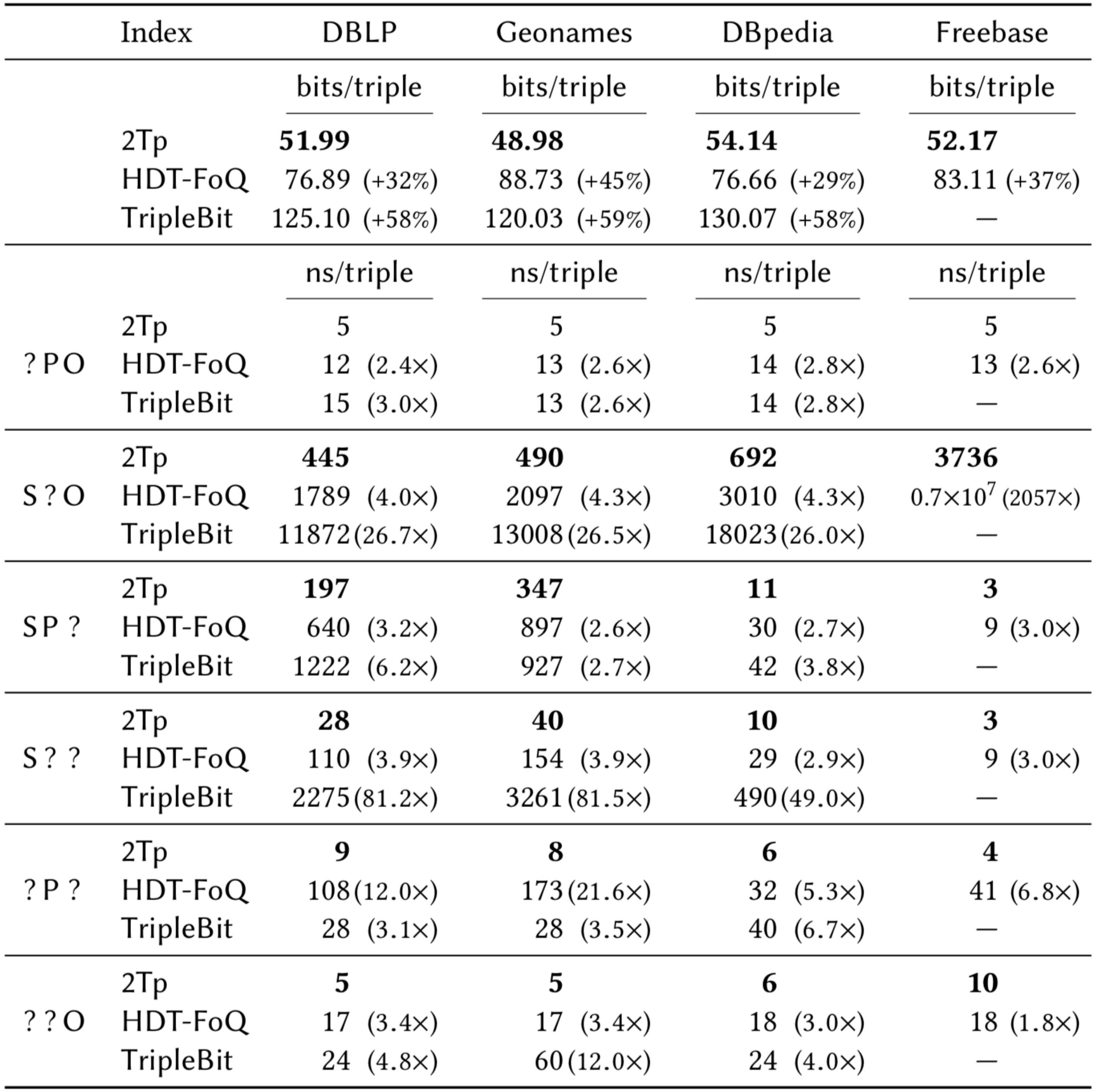
Experiments: our solutions



	Index	DBLP	Geonames	DBpedia	Freebase
		bits/triple	bits/triple	bits/triple	bits/triple
	3T	75.24 (+31%)	71.59 (+32%)	80.64 (+33%)	74.20 (+30%)
	CC	63.54 (+18%)	67.04 (+27%)	66.91 (+19%)	70.46 (+26%)
	2To	56.46 (+8%)	53.23 (+8%)	57.51 (+6%)	55.72 (+6%)
	2Tp	**51.99**	**48.98**	**54.14**	**52.17**
		ns/triple	ns/triple	ns/triple	ns/triple
S P O	*all*	203	221	353	521
S P ?	*all*	197	347	11	3
S ? ?	*all*	28	40	10	3
? ? ?	*all*	11	13	9	9
S ? O	3T,CC	2490 (5.6×)	3767 (7.7×)	1833 (2.6×)	6547 (1.8×)
	2To,2Tp	**445**	**490**	**692**	**3736**
? P O	3T,2To,2Tp	**5**	**5**	**5**	**5**
	CC	12 (2.4×)	15 (3.0×)	16 (3.2×)	14 (2.8×)
? ? O	3T,CC	12 (2.4×)	12 (2.4×)	12 (2.4×)	10 (2.0×)
	2To	**5**	**5**	**5**	**5**
	2Tp	5 (1.0×)	5 (1.0×)	6 (1.2×)	10 (2.0×)
? P ?	3T,2Tp	**9**	**8**	**6**	**6**
	CC	21 (2.3×)	36 (4.5×)	30 (5.0×)	29 (4.8×)
	2To	81 (9.0×)	138 (17.2×)	22 (3.7×)	18 (3.0×)

Overall, 2Tp offers the best space/time tradeoff.

Experiments: overall comparison



	Index	DBLP	Geonames	DBpedia	Freebase
		bits/triple	bits/triple	bits/triple	bits/triple
	2Tp	**51.99**	**48.98**	**54.14**	**52.17**
	HDT-FoQ	76.89 (+32%)	88.73 (+45%)	76.66 (+29%)	83.11 (+37%)
	TripleBit	125.10 (+58%)	120.03 (+59%)	130.07 (+58%)	—
		ns/triple	ns/triple	ns/triple	ns/triple
? P O	2Tp	5	5	5	5
	HDT-FoQ	12 (2.4×)	13 (2.6×)	14 (2.8×)	13 (2.6×)
	TripleBit	15 (3.0×)	13 (2.6×)	14 (2.8×)	—
S ? O	2Tp	**445**	**490**	**692**	**3736**
	HDT-FoQ	1789 (4.0×)	2097 (4.3×)	3010 (4.3×)	0.7×10^7 (2057×)
	TripleBit	11872 (26.7×)	13008 (26.5×)	18023 (26.0×)	—
S P ?	2Tp	**197**	**347**	**11**	**3**
	HDT-FoQ	640 (3.2×)	897 (2.6×)	30 (2.7×)	9 (3.0×)
	TripleBit	1222 (6.2×)	927 (2.7×)	42 (3.8×)	—
S ? ?	2Tp	**28**	**40**	**10**	**3**
	HDT-FoQ	110 (3.9×)	154 (3.9×)	29 (2.9×)	9 (3.0×)
	TripleBit	2275 (81.2×)	3261 (81.5×)	490 (49.0×)	—
? P ?	2Tp	**9**	**8**	**6**	**4**
	HDT-FoQ	108 (12.0×)	173 (21.6×)	32 (5.3×)	41 (6.8×)
	TripleBit	28 (3.1×)	28 (3.5×)	40 (6.7×)	—
? ? O	2Tp	**5**	**5**	**6**	**10**
	HDT-FoQ	17 (3.4×)	17 (3.4×)	18 (3.0×)	18 (1.8×)
	TripleBit	24 (4.8×)	60 (12.0×)	24 (4.0×)	—

Our selected trade-off configuration substantially outperforms the tested competitors in both space and time.

Conclusions

The *triple indexing problem with pattern matching* can be solved efficiently in both time and space regards.

Our solution — the **permuted trie index** — achieves substantial performance improvement against the best previous solutions.

Cross-compression
Permutation-elimination

Paper available at
https://arxiv.org/abs/1904.07619

C++ code available at
https://github.com/jermp/rdf_indexes

Thanks for your attention, time, patience!

Any questions?