Full-text Support for Publish/Subscribe Ontology Systems

Lefteris Zervakis, Christos Tryfonopoulos,

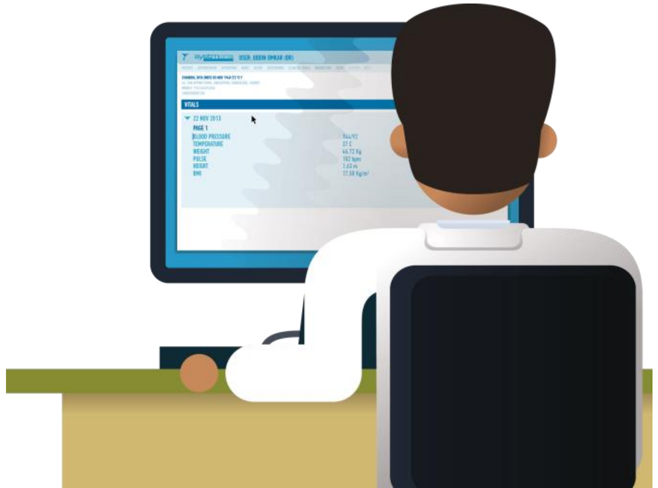
Spiros Skiadopoulos, and Manolis Koubarakis

University of the Peloponnese

National and Kapodistrian University of Athens

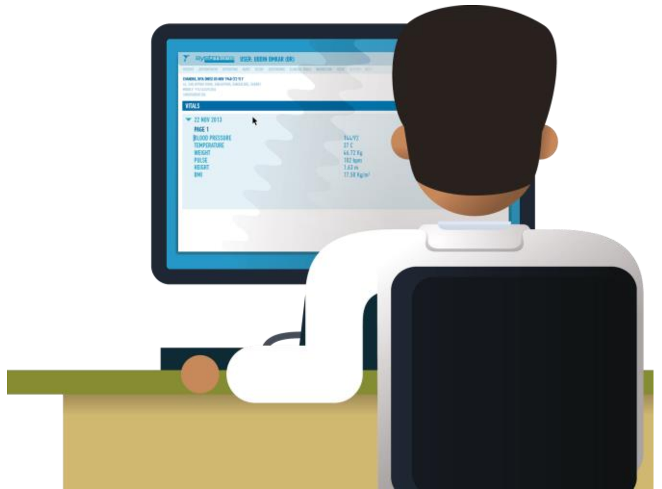
User Information Needs

- User interests
- Up to date
- Two information discovery paradigms
  - information pull
  - information push



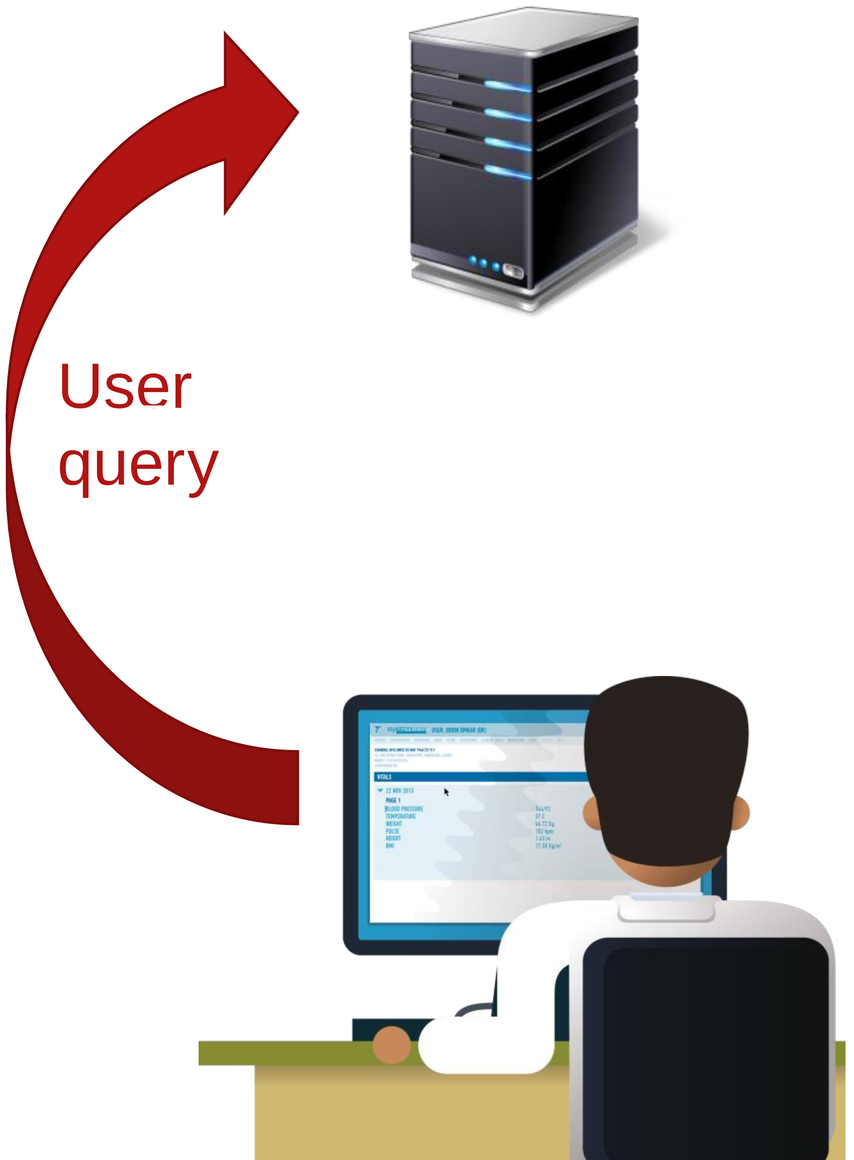
The Information Pull Paradigm

- One-time queries
  - document indexing
- Content updates
- Recurring searches
- Cognitive overload!



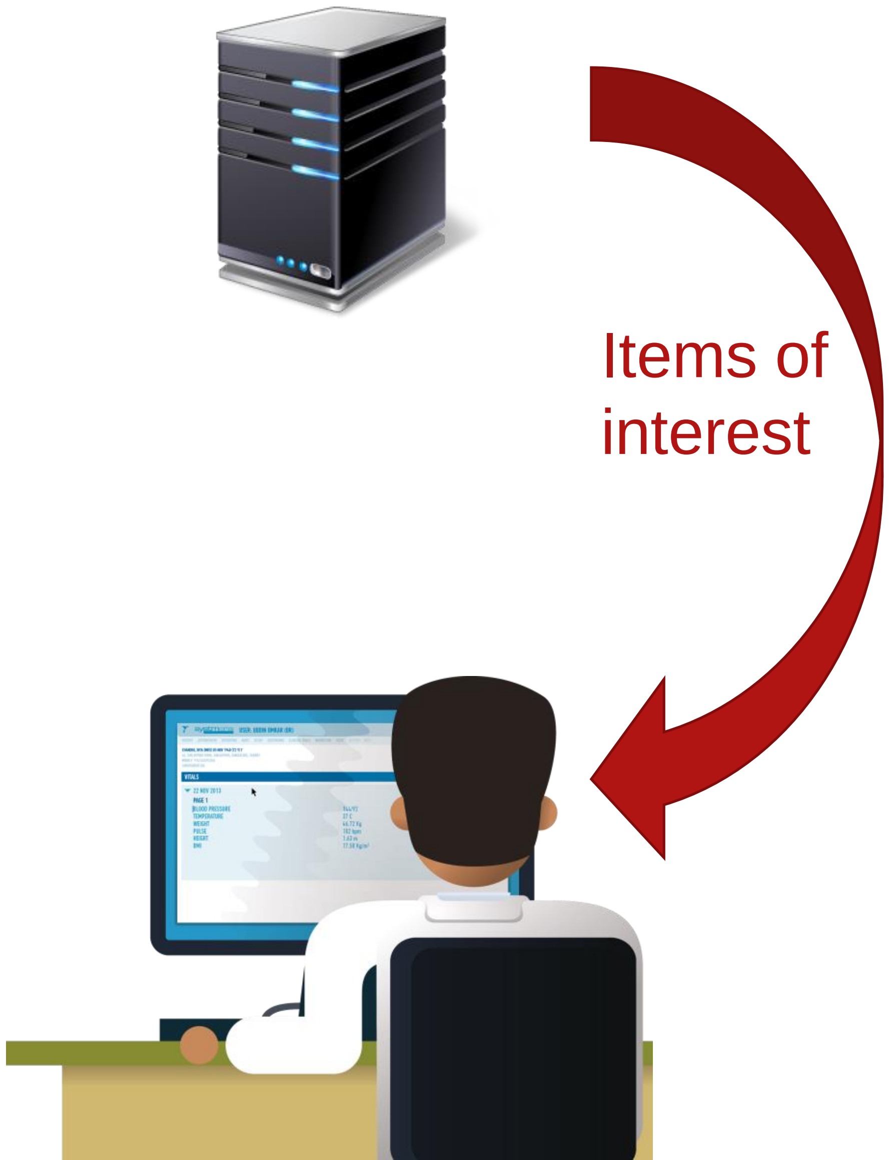
The Information Pull Paradigm

- One-time queries
  - document indexing
- Content updates
- Recurring searches
- Cognitive overload!



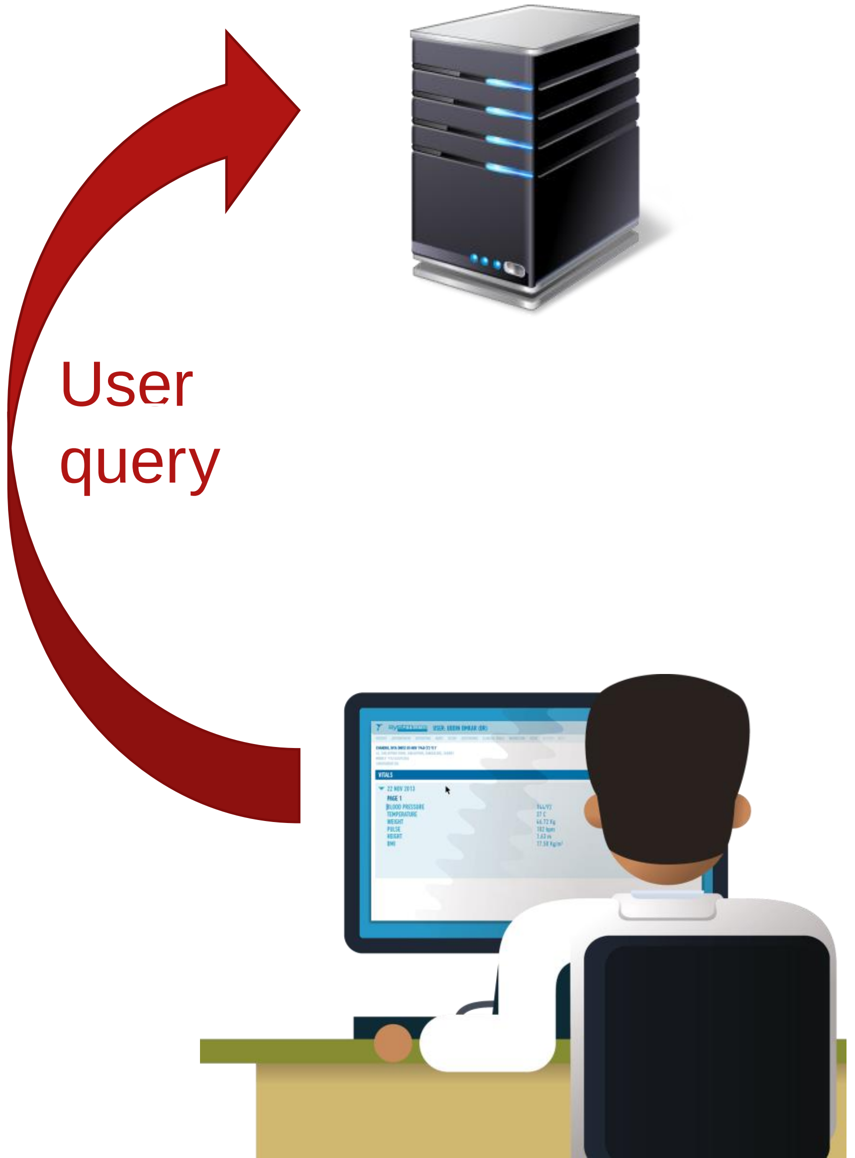
The Information Pull Paradigm

- One-time queries
  - document indexing
- Content updates
- Recurring searches
- Cognitive overload!



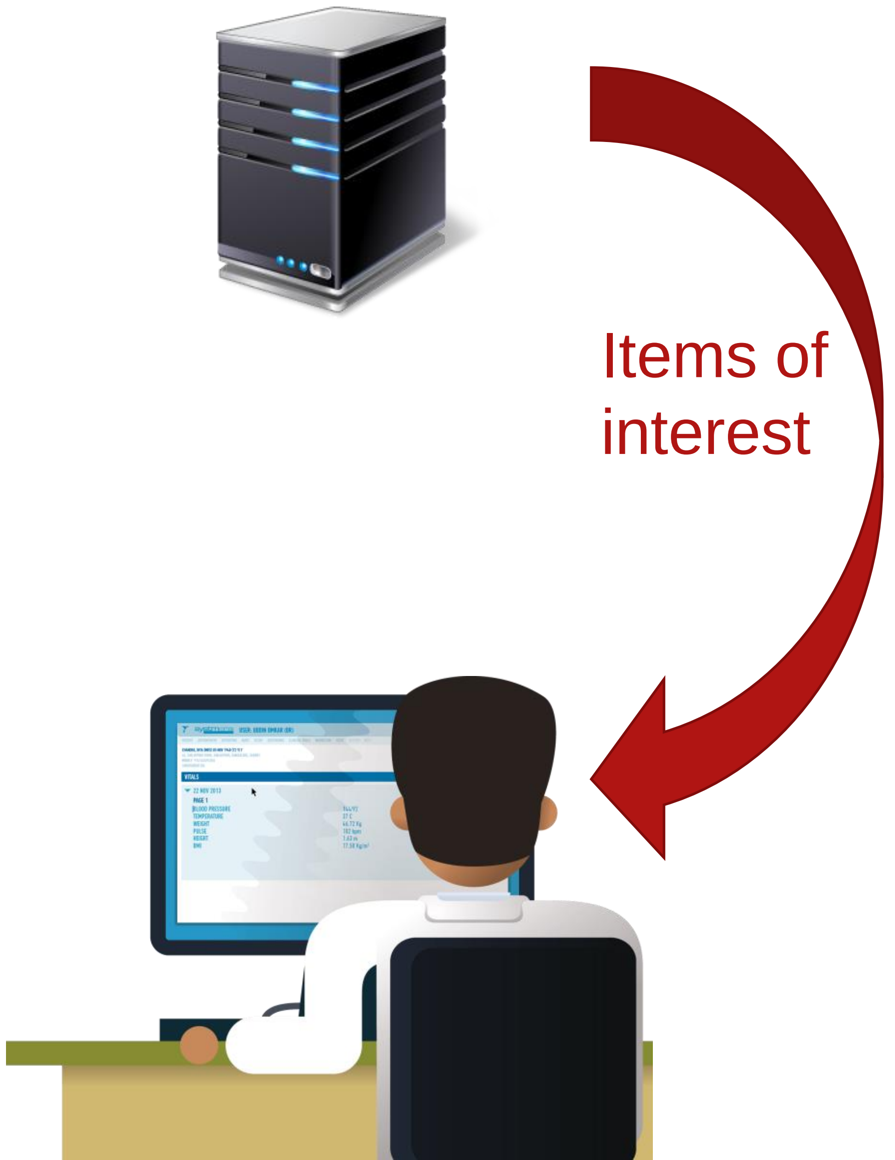
The Information Pull Paradigm

- One-time queries
  - document indexing
- Content updates
- Recurring searches
- Cognitive overload!



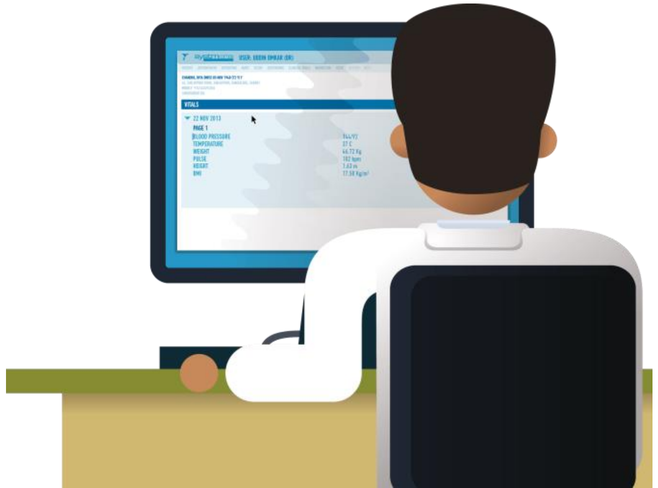
The Information Pull Paradigm

- One-time queries
  - document indexing
- Content updates
- Recurring searches
- Cognitive overload!



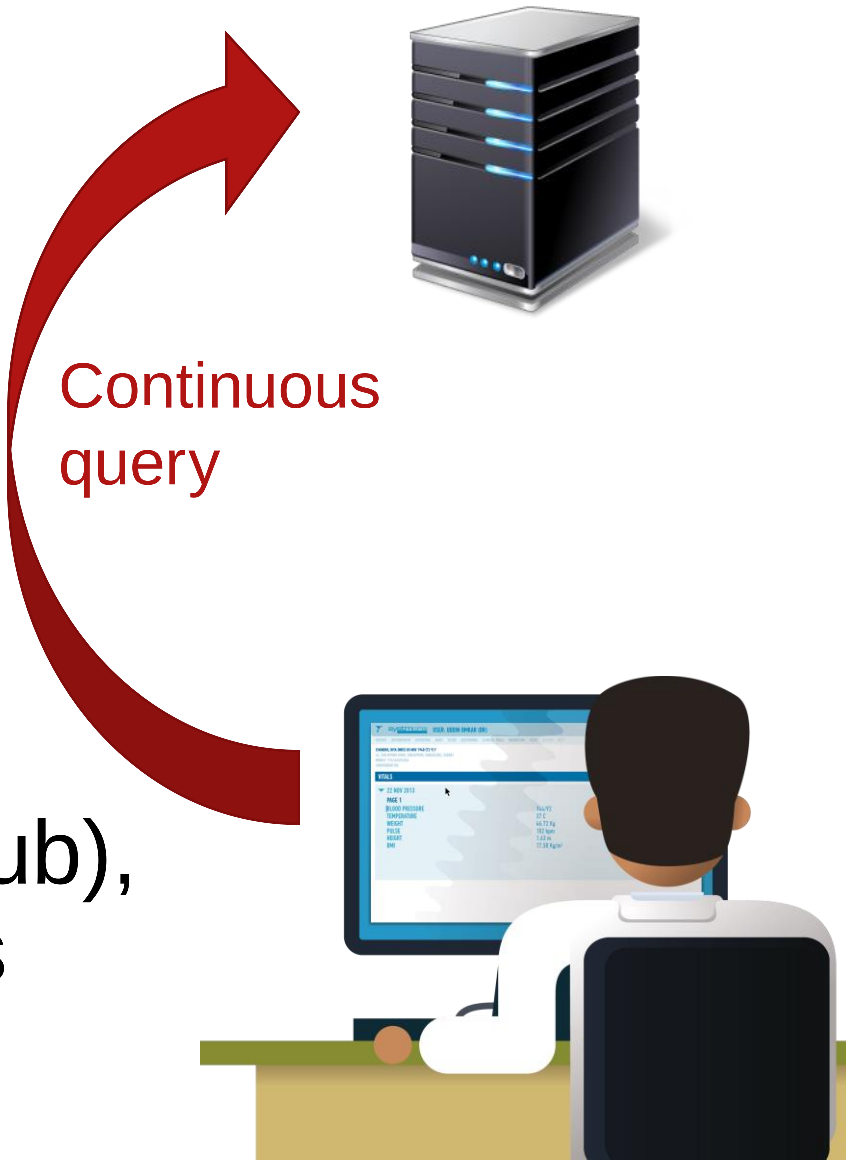
The Information Push Paradigm

- Continuous queries
- Filtering of new content
  - query indexing
- Notifications
- Push systems
  - publish/subscribe (pub/sub), alerting, filtering systems



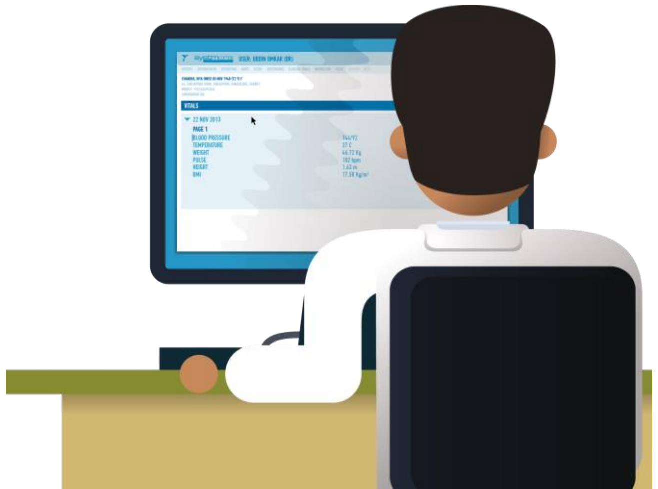
The Information Push Paradigm

- Continuous queries
- Filtering of new content
  - query indexing
- Notifications
- Push systems
  - publish/subscribe (pub/sub), alerting, filtering systems



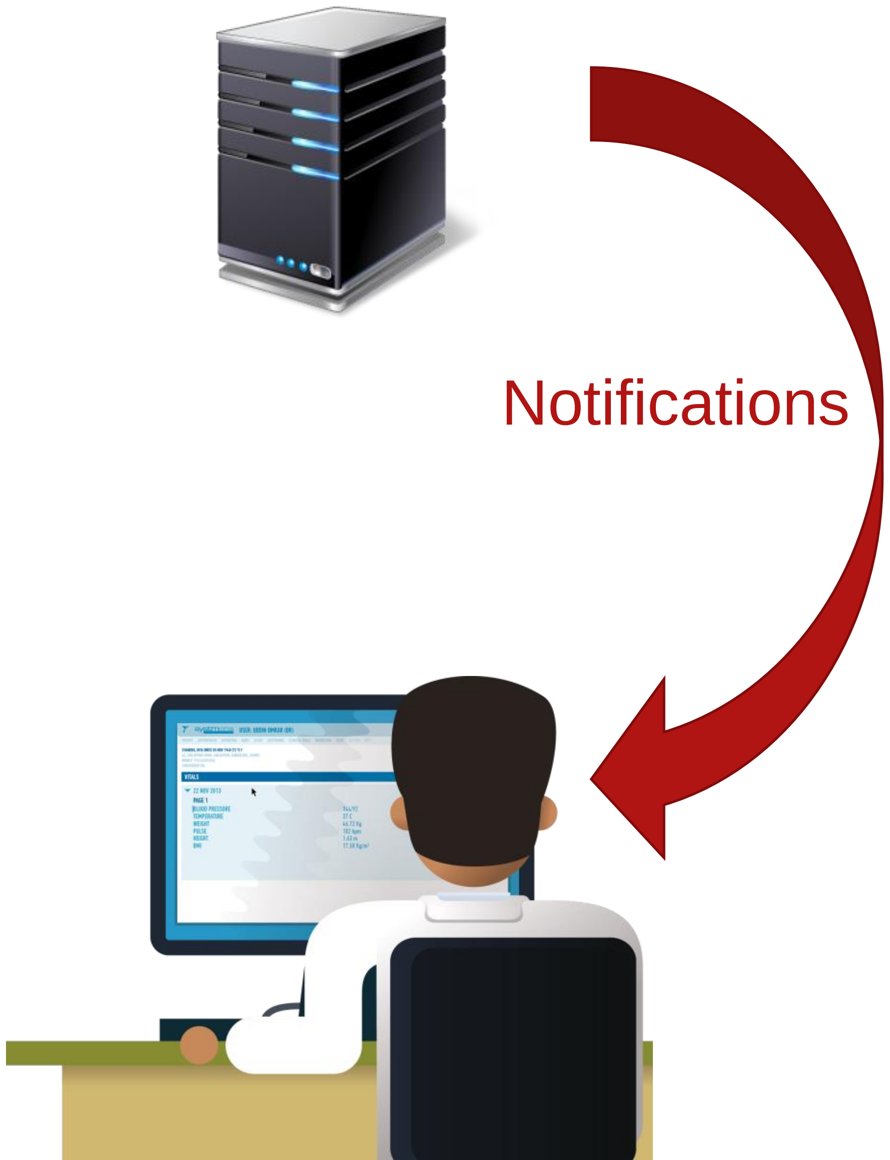
The Information Push Paradigm

- Continuous queries
- Filtering of new content
  - query indexing
- Notifications
- Push systems
  - publish/subscribe (pub/sub), alerting, filtering systems



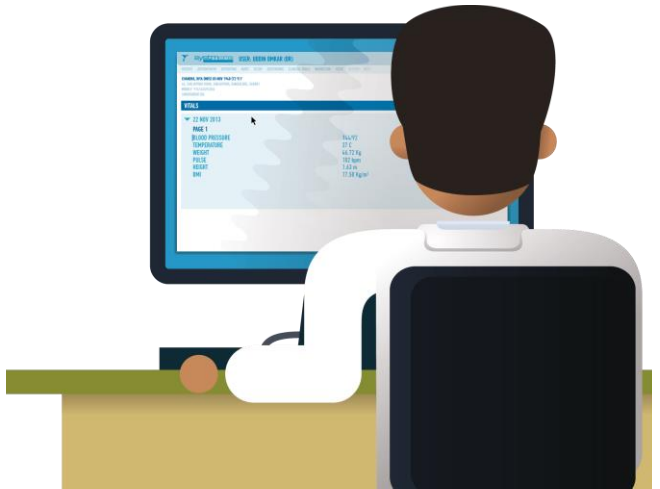
The Information Push Paradigm

- Continuous queries
- Filtering of new content
  - query indexing
- Notifications
- Push systems
  - publish/subscribe (pub/sub), alerting, filtering systems



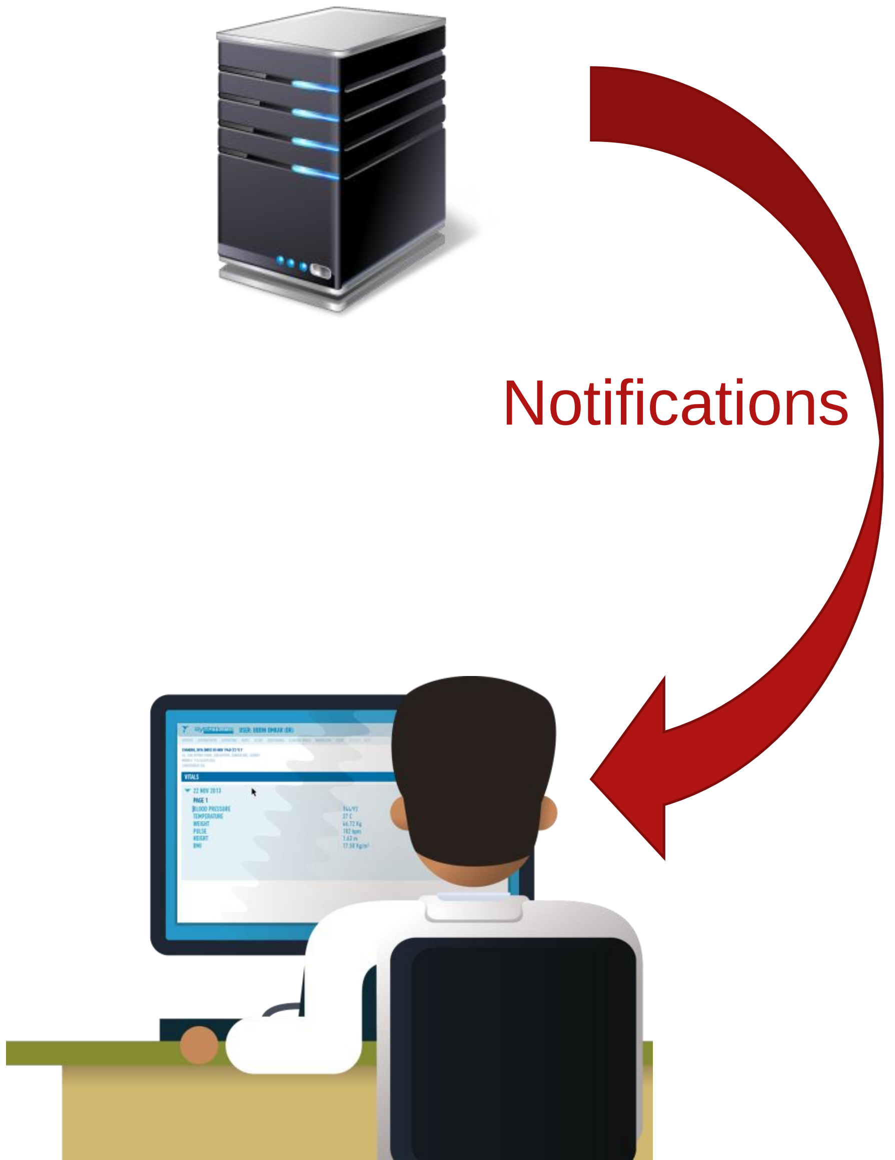
The Information Push Paradigm

- Continuous queries
- Filtering of new content
  - query indexing
- Notifications
- Push systems
  - publish/subscribe (pub/sub), alerting, filtering systems



The Information Push Paradigm

- Continuous queries
- Filtering of new content
  - query indexing
- Notifications
- Push systems
  - publish/subscribe (pub/sub), alerting, filtering systems



Ontology-based Publish/Subscribe

- Enhanced semantics
- Subscriptions are SPARQL queries
- Publications are sets of RDF triples

Current State of the Art

- Structural filtering (S-ToPSS, G-ToPSS)
- Structural filtering + arithmetic/string operations (iBroker)
- No structural + full-text filtering
  - contrary to information pull systems

Applications

- Ontology-enabled
  - news alerts (RSS feeds)
  - digital libraries
- Curation/monitoring tool for linked datasets
- Complement LOD platforms
  - structural/textual notifications

Our Contribution (1/2)

- Extend SPARQL with full-text pub/sub
  - Boolean, word proximity, phrase operators

```
SELECT ?publication
WHERE {?publication type article.
            ?publication title ?title.
            ?publication body ?body.
FILTER ftcontains(?title, "olympic" ftAND "games")
FILTER ftcontains(?body, "olympic" ftAND "games" ftNEAR[0,2] "rio")}
```

Our Contribution (1/2)

- Extend SPARQL with full-text pub/sub
  - Boolean, word proximity, phrase operators

```
SELECT ?publication
WHERE {?publication type article.
        ?publication title ?title.
        ?publication body ?body.
FILTER ftcontains(?title, "olympic" ftAND "games")
FILTER ftcontains(?body, "olympic" ftAND "games" ftNEAR[0,2] "rio")}
```

Our Contribution (1/2)

- Extend SPARQL with full-text pub/sub
  - Boolean, word proximity, phrase operators

```
SELECT ?publication
WHERE {?publication type article.
            ?publication title ?title.
            ?publication body ?body.
FILTER ftcontains(?title, "olympic" ftAND "games")
FILTER ftcontains(?body, "olympic" ftAND "games" ftNEAR[0,2] "rio")}
```

Our Contribution (1/2)

- Extend SPARQL with full-text pub/sub
  - Boolean, word proximity, phrase operators

```
SELECT ?publication
WHERE {?publication type article.
              ?publication title ?title.
              ?publication body ?body.
FILTER ftcontains(?title, "olympic" ftAND "games")
FILTER ftcontains(?body, "olympic" ftAND "games" ftNEAR[0,2] "rio")}
```

Our Contribution (2/2)

- RTF: **R**DF and **T**ext **F**iltering
  - structural filtering
  - full-text filtering
- Focus on efficiency

Our Contribution (2/2)

- RTF: **R**DF and **T**ext **F**iltering
  - structural filtering
  - full-text filtering

$\Big\} \Rightarrow$ Index in a unified way (tree-based)

- Focus on efficiency

SPARQL Query to Tuple Conjucts

Query q:

```
SELECT ?publication
WHERE {?publication type article.
                ?publication title ?title.
                ?publication body ?body.
FILTER ftcontains(?title, "olympic" ftAND "games")
FILTER ftcontains(?body, "olympic" ftAND "games"
ftNEAR[0,2] "rio")}
```

SPARQL Query to Tuple Conjucts

Query q:

SELECT ?publication

WHERE {$\boxed{\text{?publication type article.}}$

?publication title ?title.

?publication body ?body.

FILTER ftcontains(?title, "olympic" ftAND "games")

FILTER ftcontains(?body, "olympic" ftAND "games" ftNEAR$_{[0,2]}$ "rio")}

SPARQL Query to Tuple Conjucts

$q.t_1$ = (? publication , type , article)

Query q:

```
SELECT ?publication
WHERE {?publication type article.
       ?publication title ?title.
       ?publication body ?body.
FILTER ftcontains(?title, "olympic" ftAND "games")
FILTER ftcontains(?body, "olympic" ftAND "games"
ftNEAR[0,2] "rio")}
```

SPARQL Query to Tuple Conjucts

Query q:

```
SELECT ?publication
WHERE {?publication type article.
       ?publication title ?title.
       ?publication body ?body.
FILTER ftcontains(?title, "olympic" ftAND "games")
FILTER ftcontains(?body, "olympic" ftAND "games"
ftNEAR[0,2] "rio")}
```

SPARQL Query to Tuple Conjucts

Query q:

SELECT ?publication

WHERE {?publication type article.

$\boxed{\text{?publication title ?title.}}$

?publication body ?body.

$\boxed{\text{FILTER ftcontains(?title, "olympic" ftAND "games")}}$

FILTER ftcontains(?body, "olympic" ftAND "games" ftNEAR$_{[0,2]}$ "rio")}

SPARQL Query to Tuple Conjucts

$q.t_2$ = (? publication , title , ?title ,
ftcontains (" olympic " ftAND " games "))

Query q:

```
SELECT ?publication
WHERE {?publication type article.
        ?publication title ?title.
        ?publication body ?body.
FILTER ftcontains(?title, "olympic" ftAND "games")
FILTER ftcontains(?body, "olympic" ftAND "games"
ftNEAR[0,2] "rio")}
```

SPARQL Query to Tuple Conjucts

Query q:

SELECT ?publication

WHERE {?publication type article.

?publication title ?title.

?publication body ?body.

FILTER ftcontains(?title, "olympic" ftAND "games")

FILTER ftcontains(?body, "olympic" ftAND "games" $\text{ftNEAR}_{[0,2]}$ "rio")}

SPARQL Query to Tuple Conjucts

$q.t_3$ = (? publication , title , ?title ,
ftcontains (" olympic " ftAND " games " ftNEAR[0,2] "rio"))

Query q

```
SELECT ?publication
WHERE {?publication type article.
            ?publication title ?title.
            ?publication body ?body.
FILTER ftcontains(?title, "olympic" ftAND "games")
FILTER ftcontains(?body, "olympic" ftAND "games"
ftNEAR[0,2] "rio")}
```

SPARQL Query to Tuple Conjucts

$q.t_1$ = (? publication , type , article)

$q.t_2$ = (? publication , title , ?title ,
ftcontains (" olympic " ftAND " games "))

q.t3 = (? publication , title , ?title ,
ftcontains (" olympic " ftAND " games " ftNEAR[0,2] "rio"))

SPARQL Query to Tuple Conjucts

$q.t_1$ = (? publication , type , article)

$q.t_2$ = (? publication , title , ?title ,
ftcontains (" olympic " ftAND " games "))

q.t3 = (? publication , title , ?title ,
ftcontains (" olympic " ftAND " games " ftNEAR[0,2] "rio"))

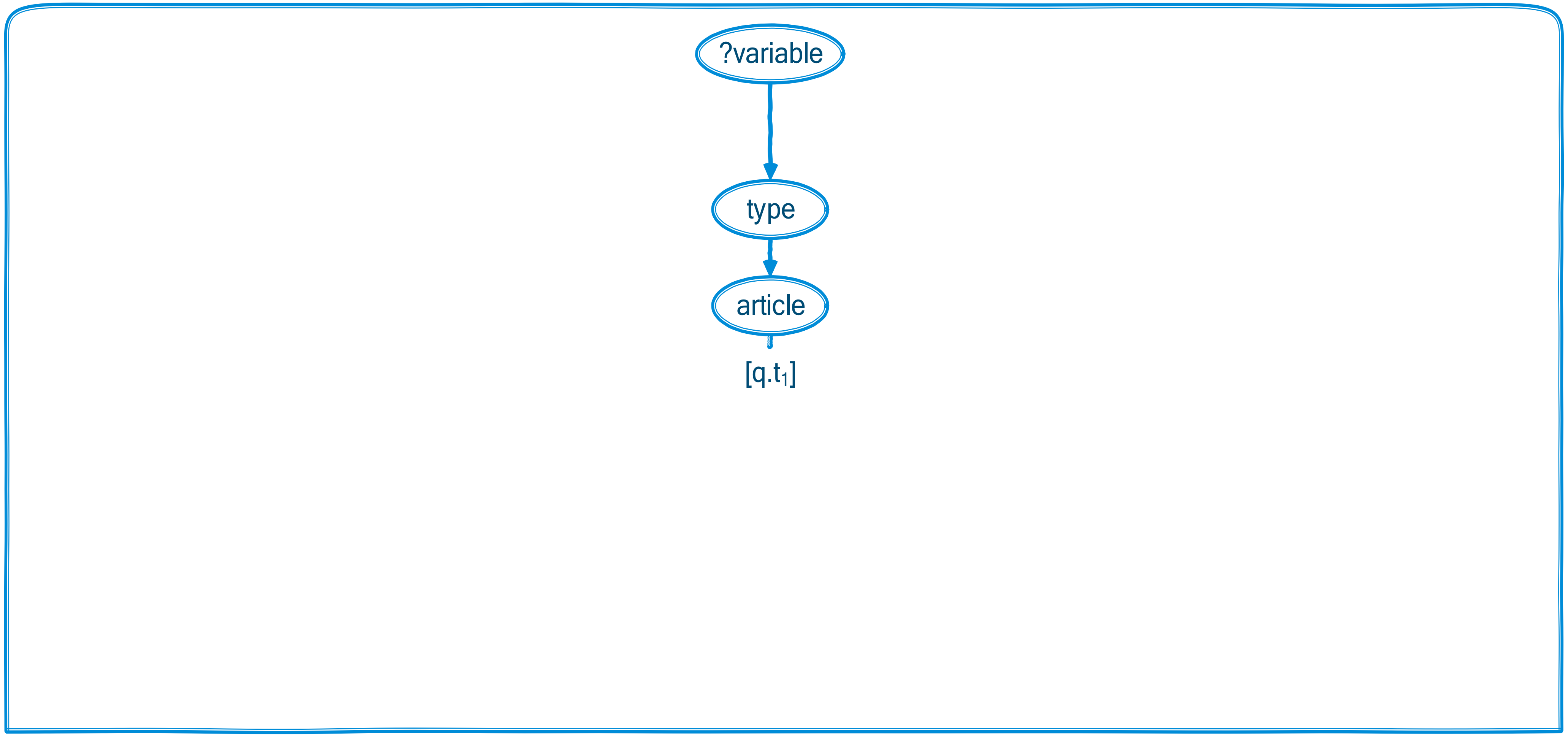
$$\mathbf{q = q.t_1 \wedge q.t_2 \wedge q.t_3}$$

RTF Tree-based Indexing

$q.t_1$ = (? publication , type , article)

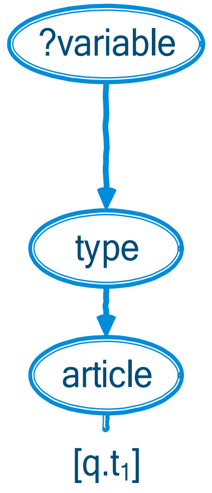
RTF Tree-based Indexing

$q.t_1$ = (? publication , type , article)



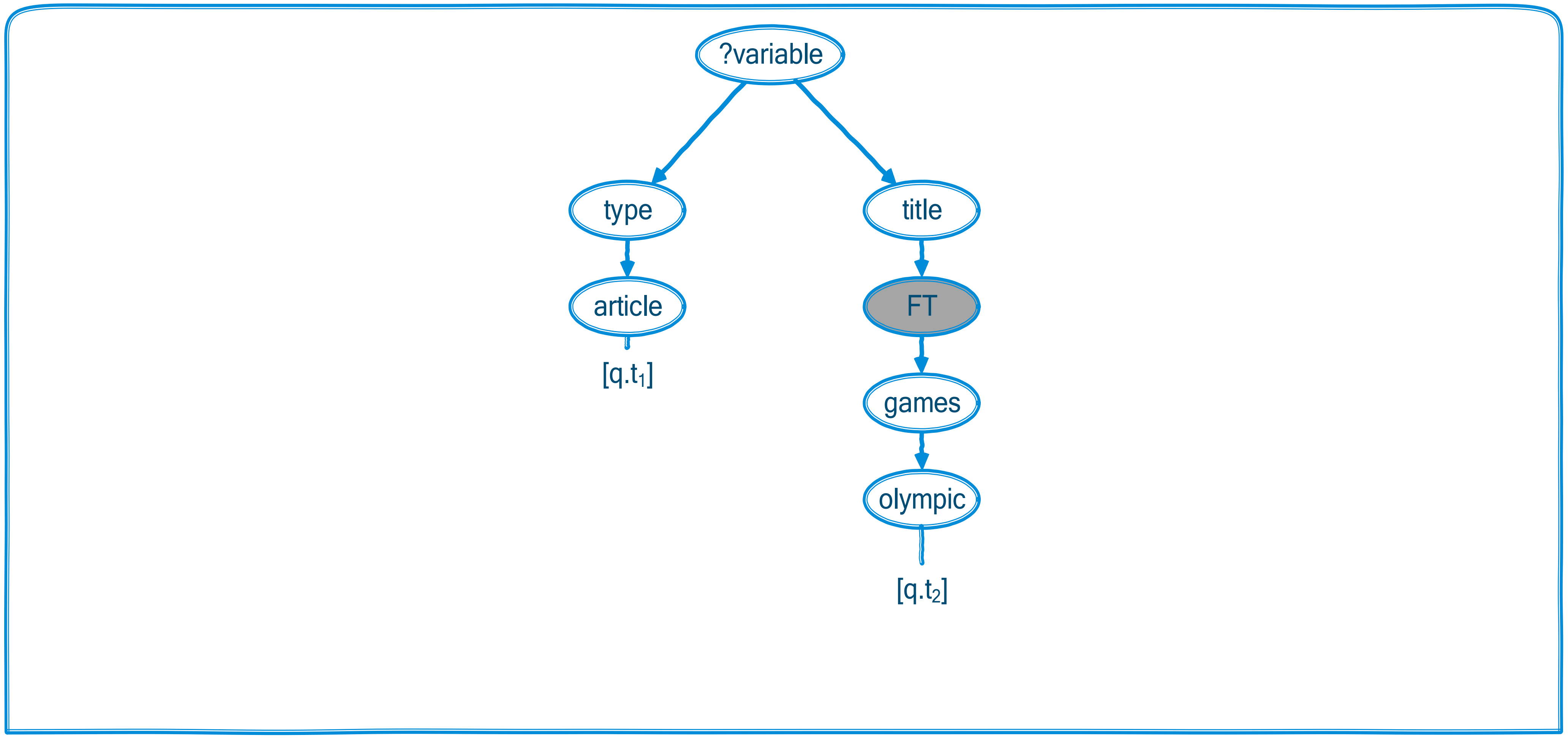
RTF Tree-based Indexing

q.t_2 = (? publication , title , ?title , ftcontains (" olympic " ftAND " games "))



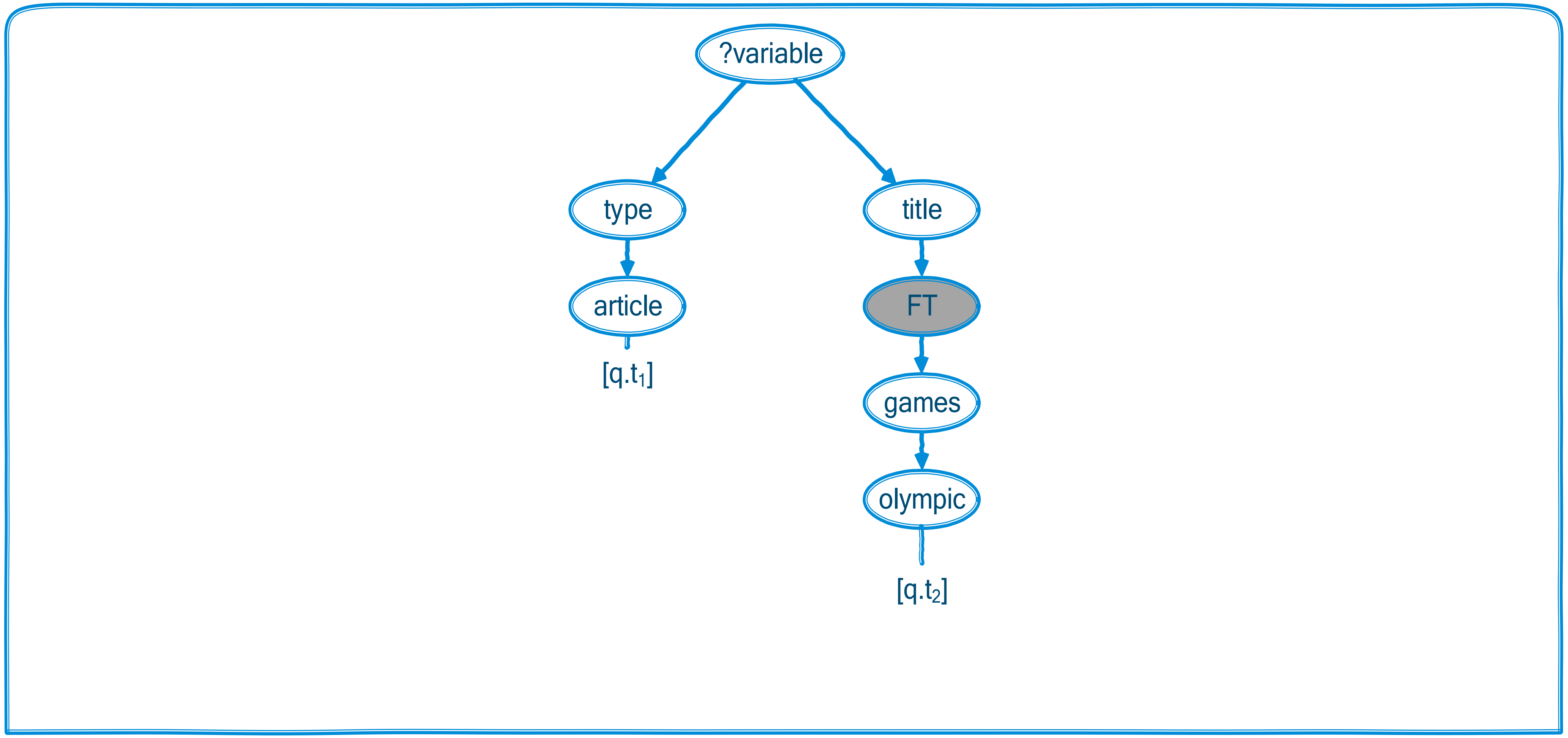
RTF Tree-based Indexing

$q.t_2$ = (? publication , title , ?title , ftcontains (" olympic " ftAND " games "))



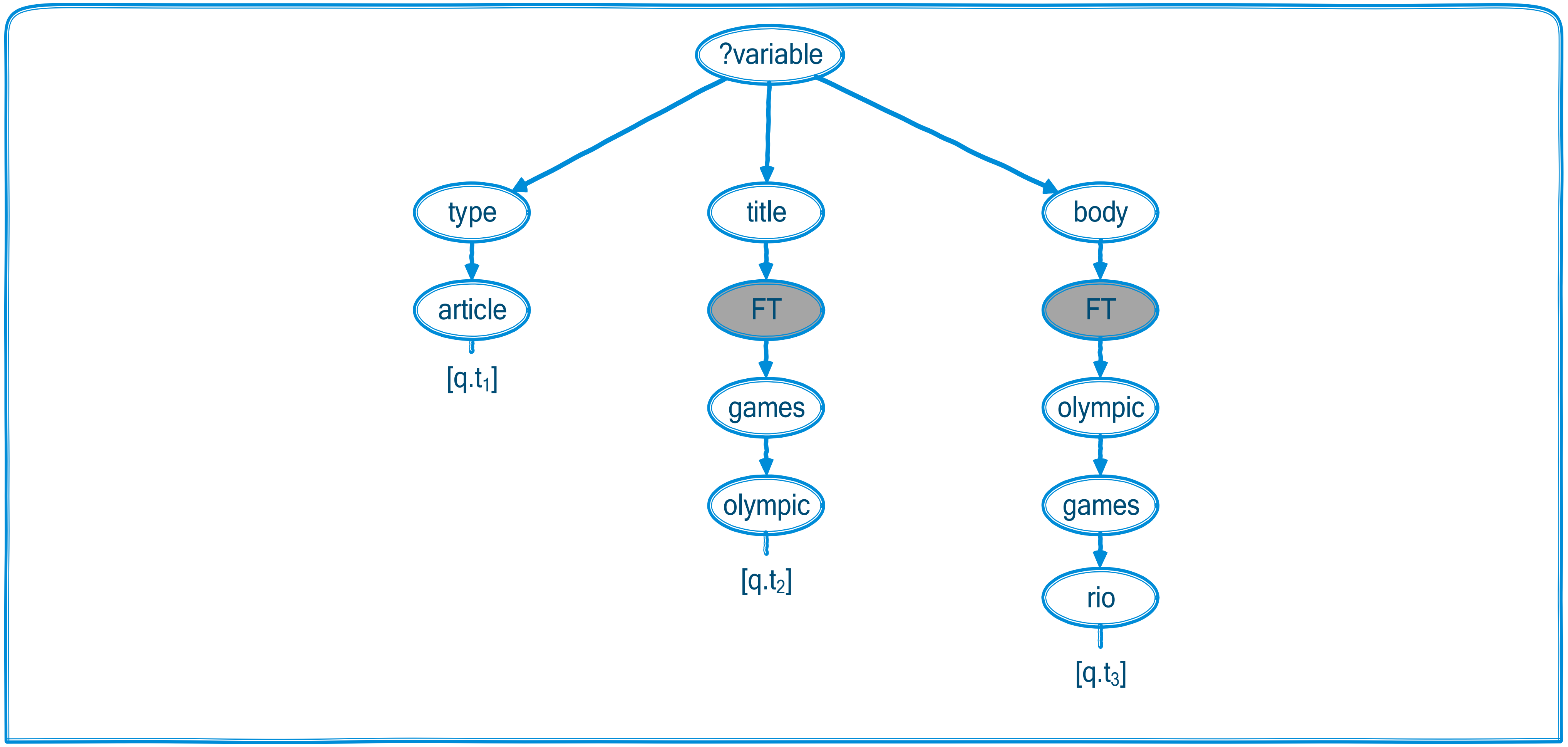
RTF Tree-based Indexing

$q.t_3$ = (? publication , body , ?body , ftcontains (" olympic " ftAND " games " ftNEAR [0,2] " rio ")



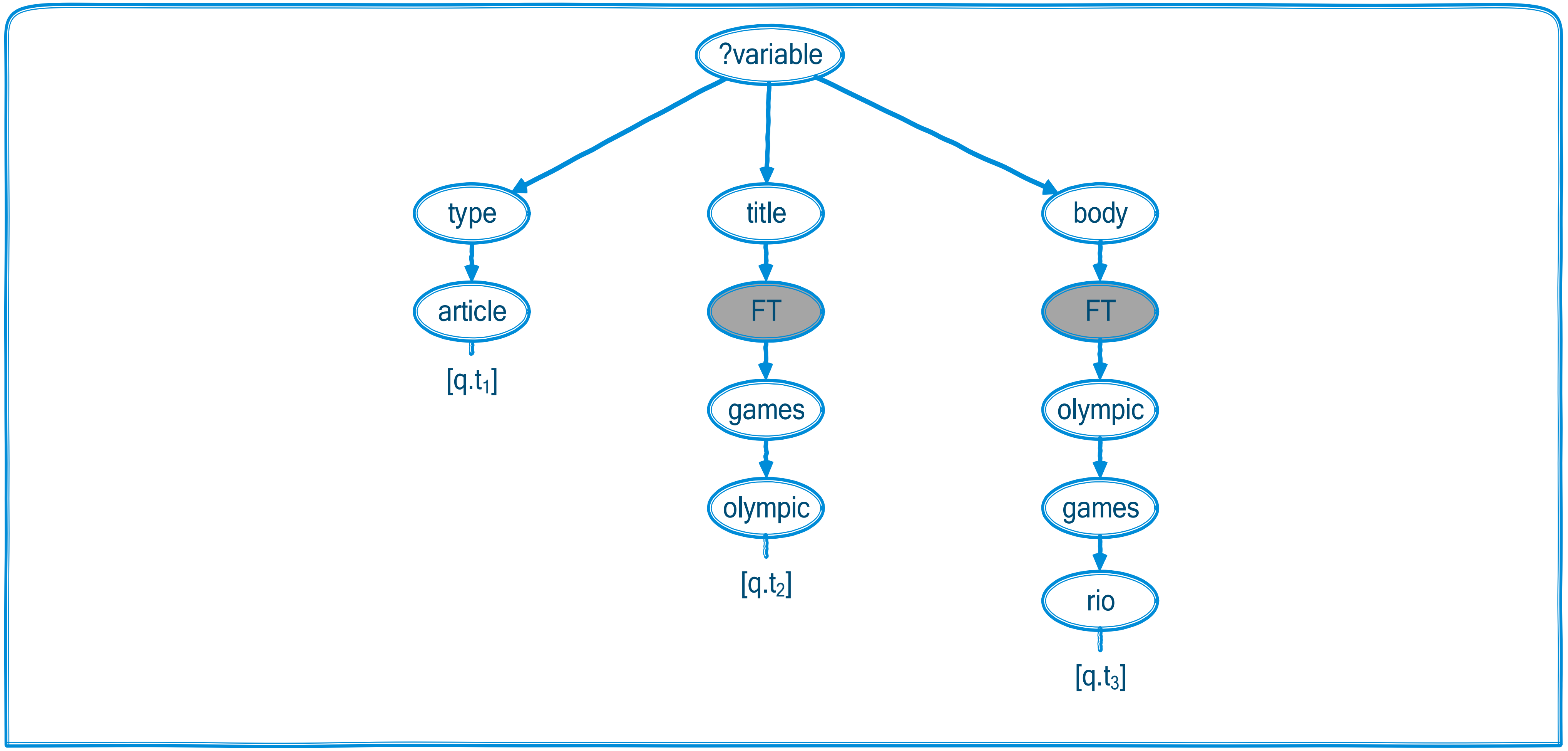
RTF Tree-based Indexing

$q.t_3$ = (? publication , body , ?body , ftcontains (" olympic " ftAND " games " ftNEAR [0,2] " rio ")



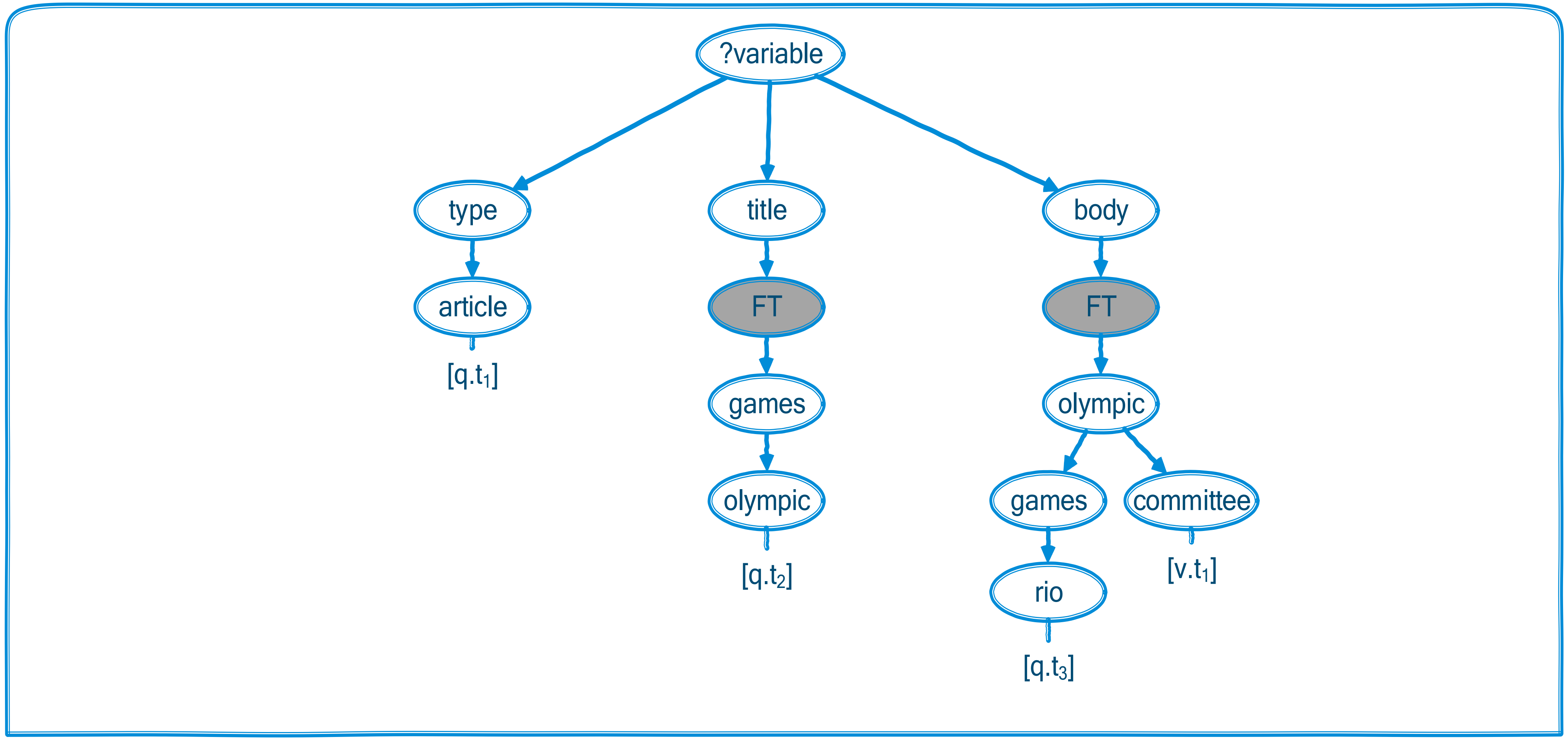
RTF Tree-based Indexing

$v.t_1$ = (? publication , body , ?body , ftcontains (" olympic " ftAND " committee ")



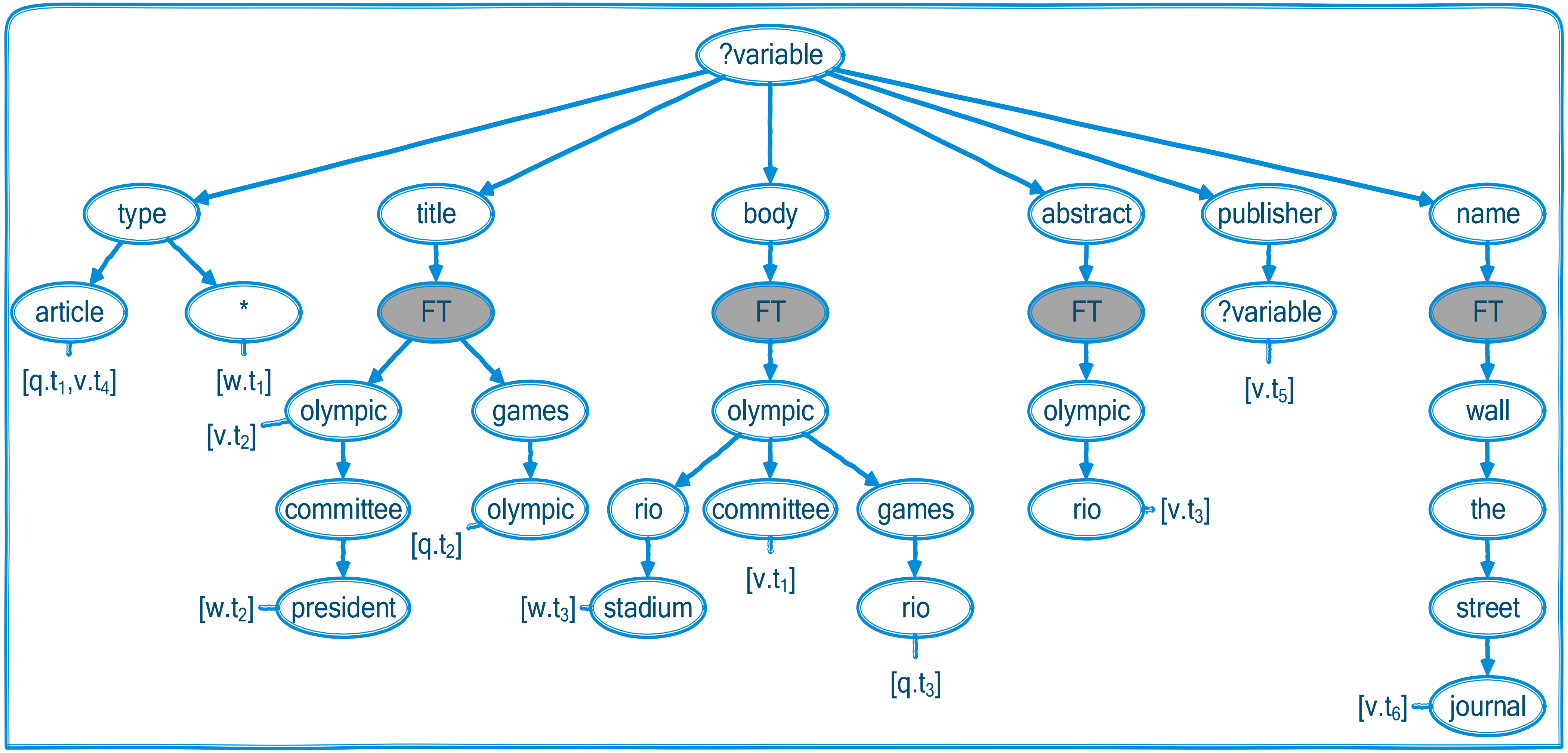
RTF Tree-based Indexing

$v.t_1$ = (? publication , body , ?body , ftcontains (" olympic " ftAND " committee ")



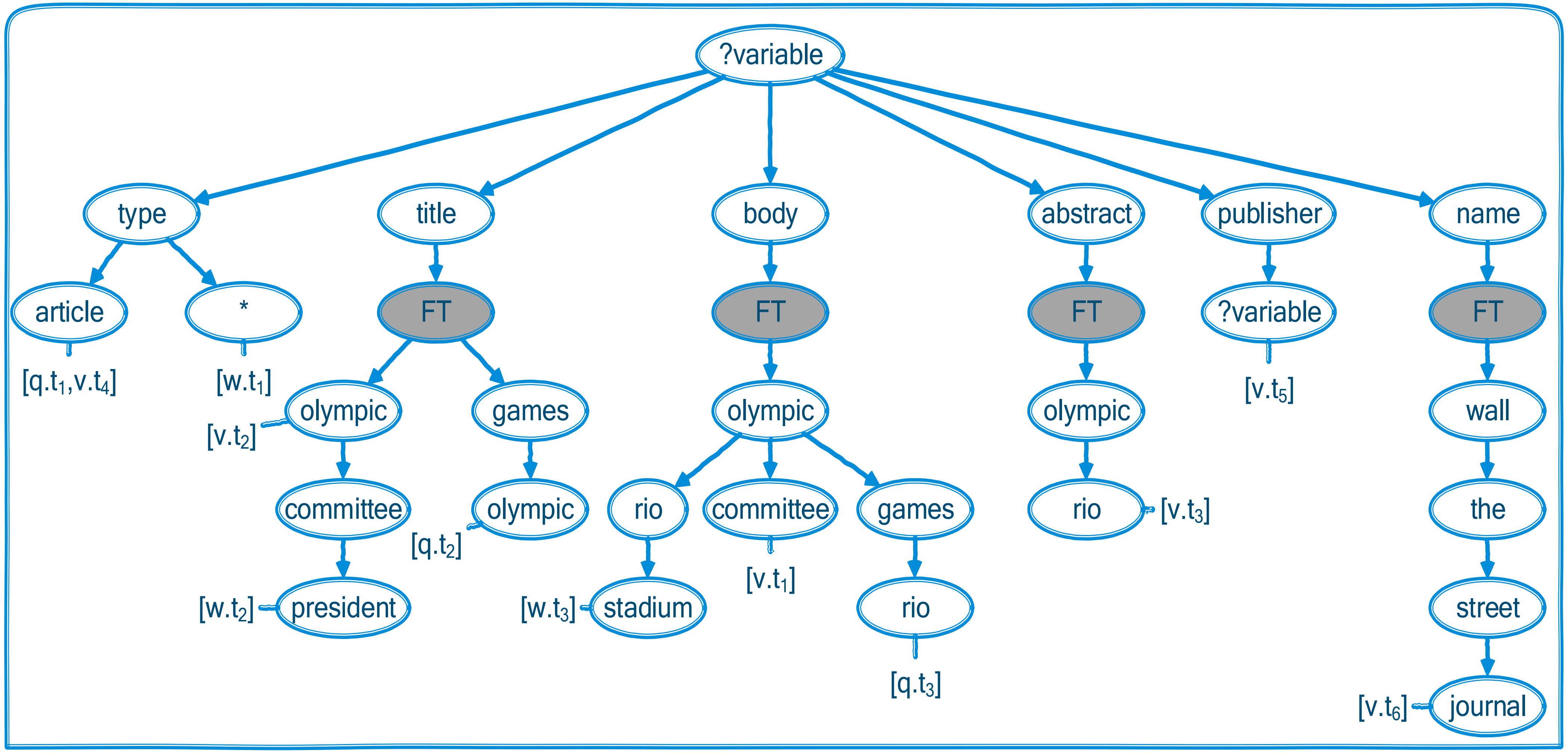
RTF Tree-based Indexing

After several tuple insertions



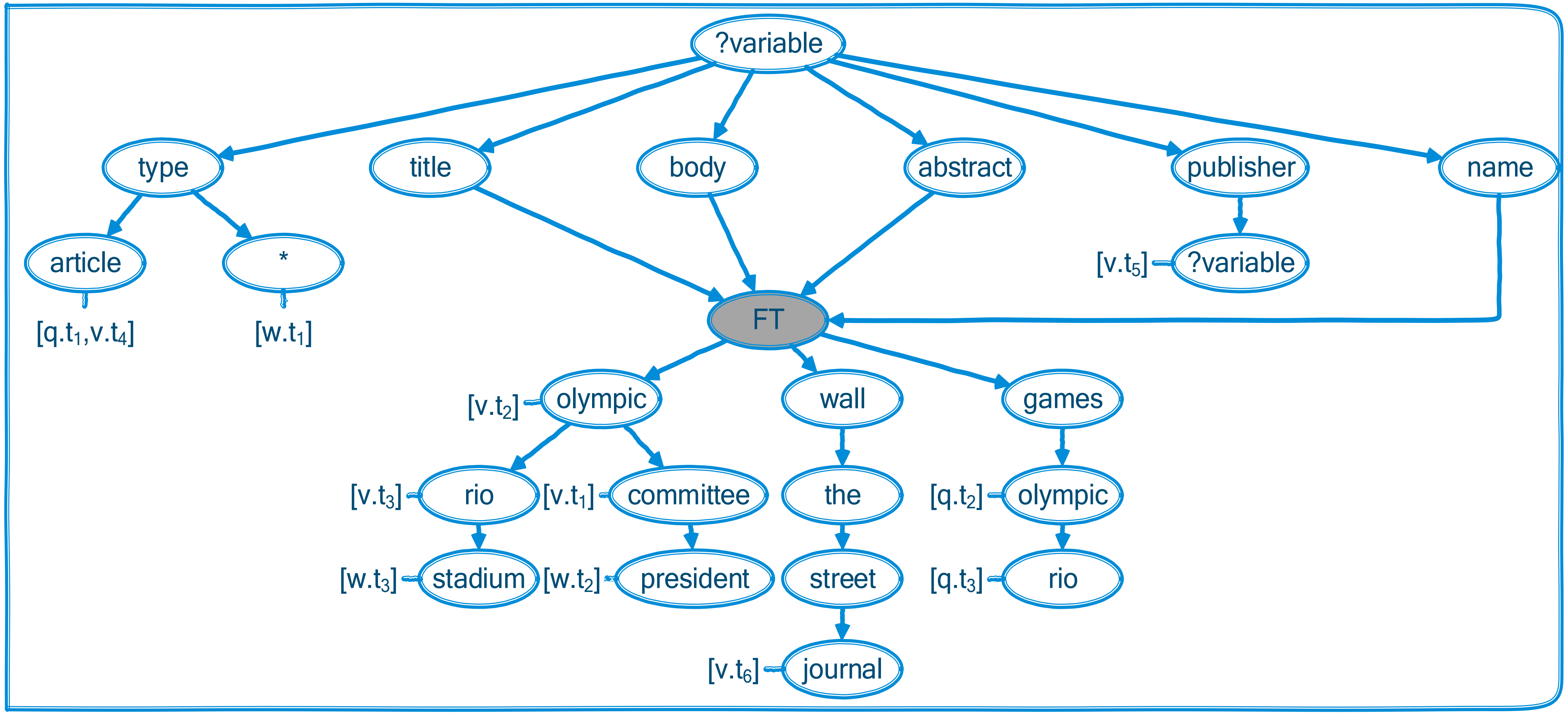
RTF Tree-based Indexing

Algorithm **RTFm** (No pun intended)



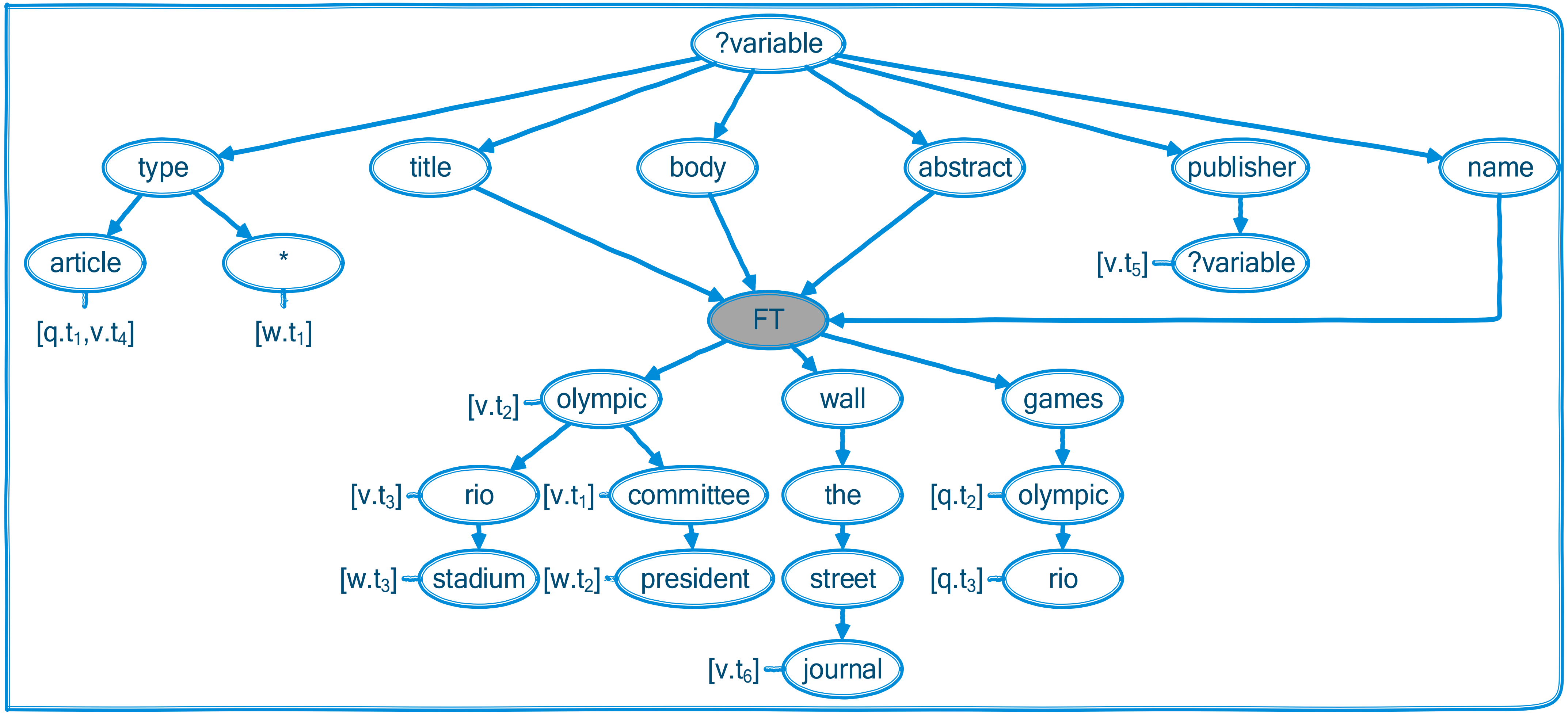
RTF Tree-based Indexing

Collapsing FT nodes



RTF Tree-based Indexing

Algorithm **RTFs**



Publication Filtering

- New publications
  - sets of RDF triples
- DFS tree traversal
- Early pruning of non-matching trees

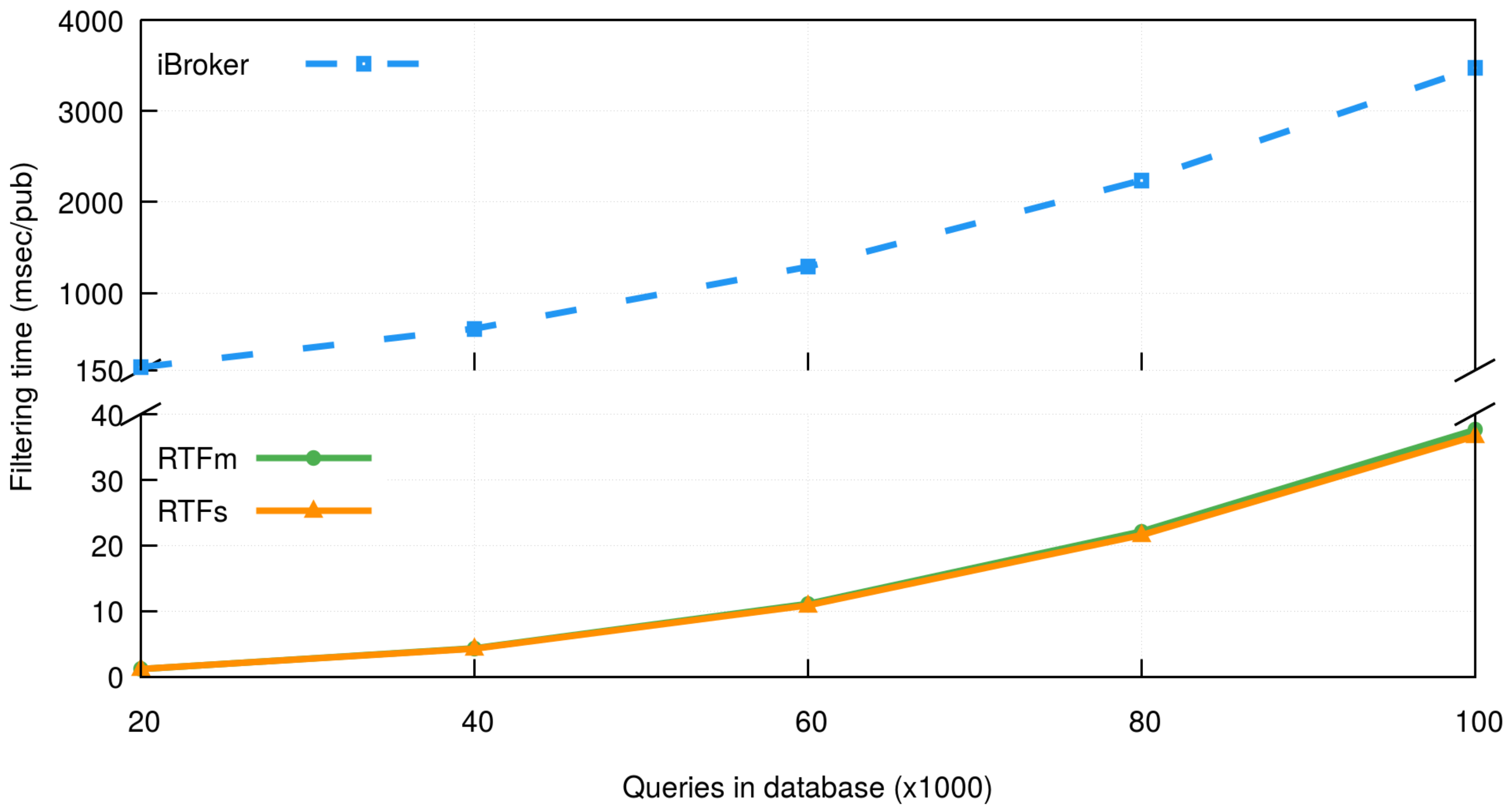
Experimental Evaluation (1/2)

- Algorithm iBroker
  - ontology pub/sub
  - structural filtering and string equality
  - inverted index
  - extended to support Boolean full-text

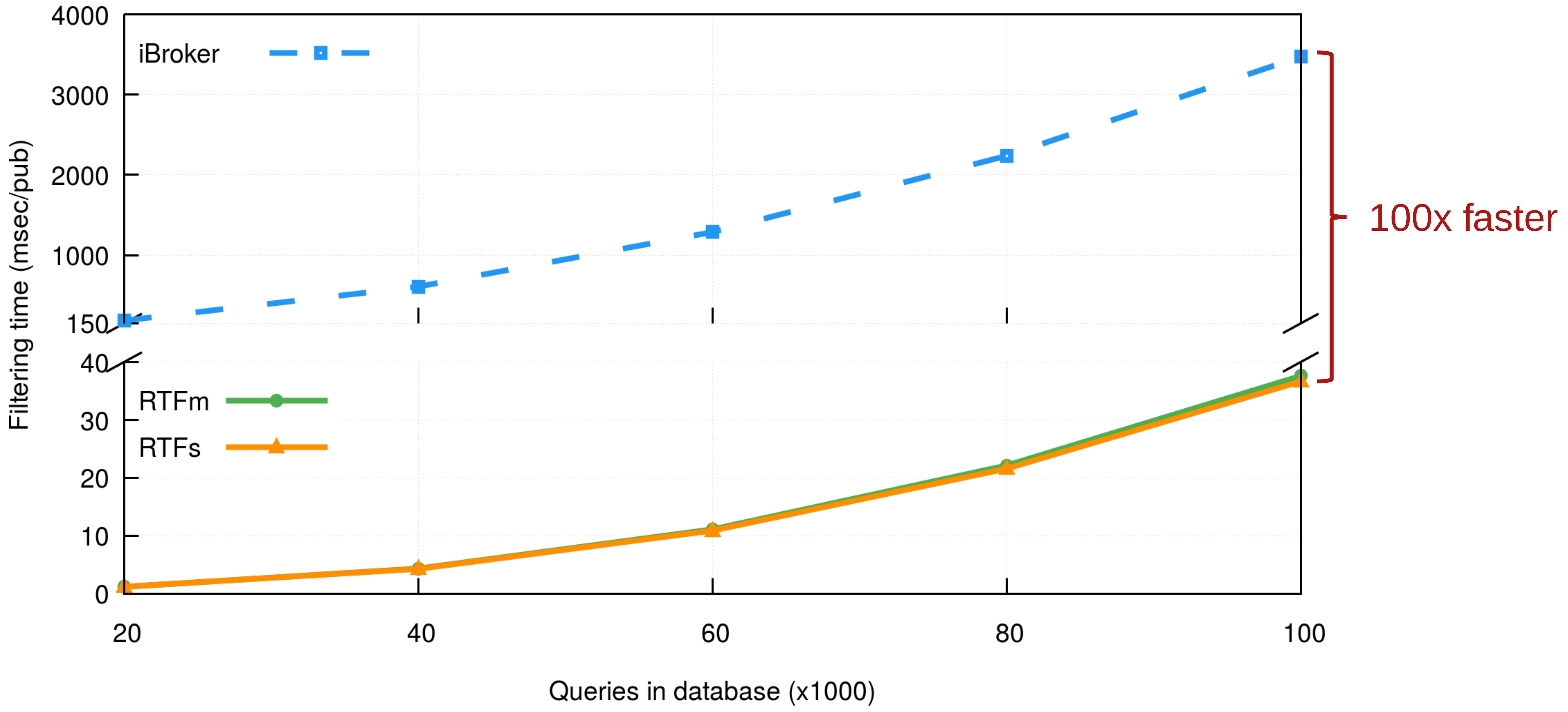
Experimental Evaluation (2/2)

- DBpedia corpus
  - 3.22M publications
  - 529 classes
  - 2.3K properties
- Synthetic continuous queries

Filtering Results



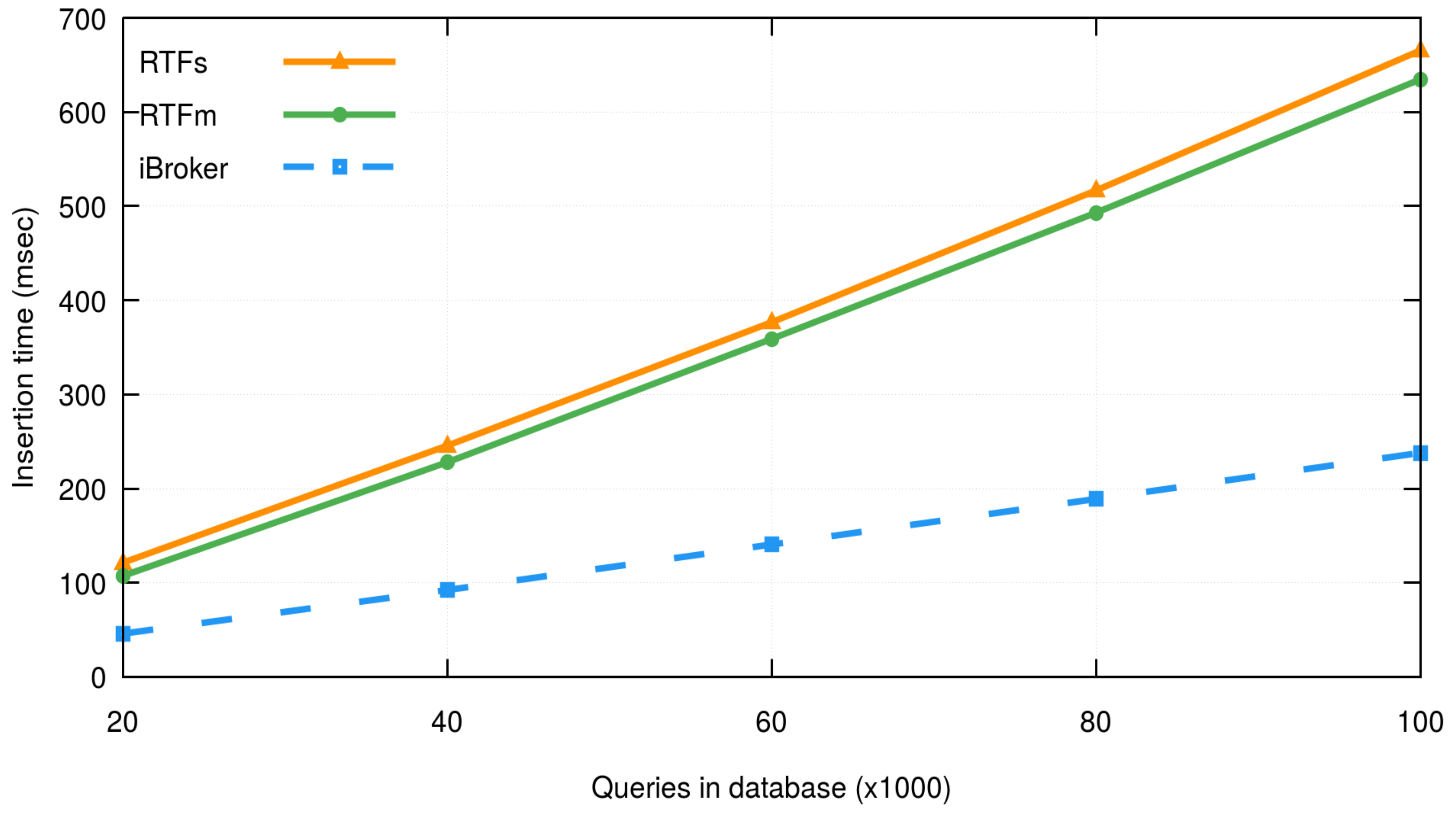
Filtering Results



More Filtering Results

- RTFs 2.5% faster than RTFm
- RTF time spent:
  - 50% for structural filtering
  - 50% for full-text filtering
- 40% less memory than iBroker

Insertion Results



Conclusions

- Extended SPARQL for Boolean full-text pub/sub
- Designed fast query indexing algorithms
  - first in the literature
  - 100x faster than competitor

- Next: support for VSM pub/sub

Thank you for your attention!

Questions?

Thank you for your attention!

Questions?