# Storing UIMA CASes in a relational database

**Georg Fette**, Martin Toepfer, Frank Puppe

# Storing UIMA CASes in a relational database

- Problems/Motivation
- Solution
- Evaluation

## CAS

- data structure storing document and annotation data
- usually stored as serialized xml files in the file system

## Type System

- data structure storing meta data about annotations
- usually stored as serialized xml files in the file system

## 1. Problem: Organization of CASes and type systems

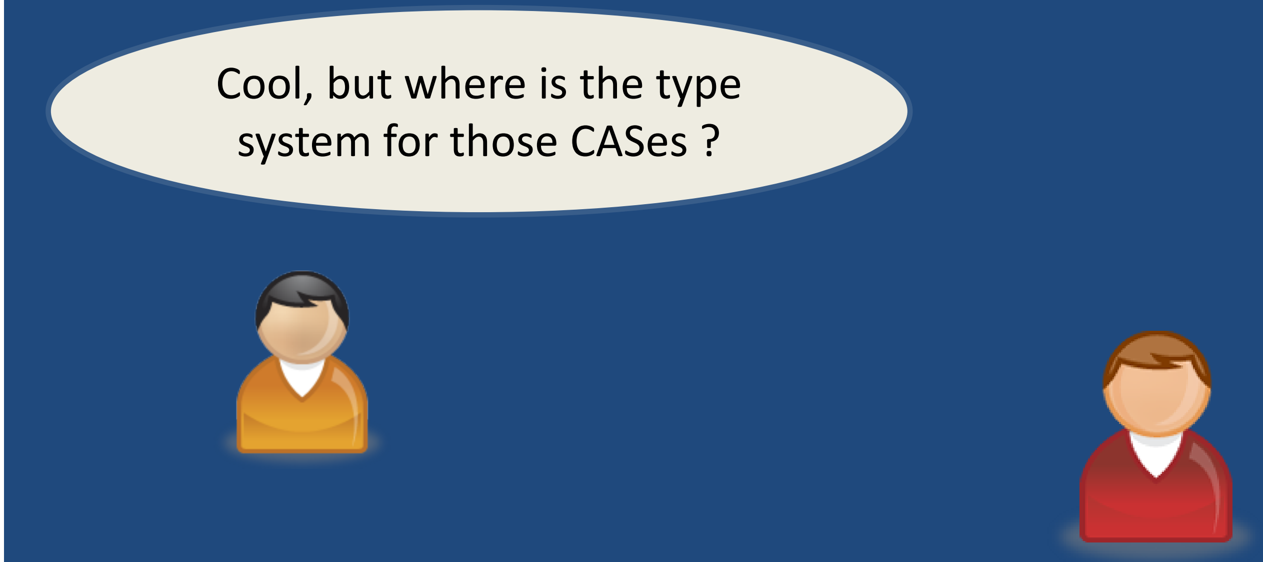
Peter, could you tell me where the CASes for your latest Named-Entity-Recognition experiments are stored?

## 1. Problem: Organization of CASes and type systems

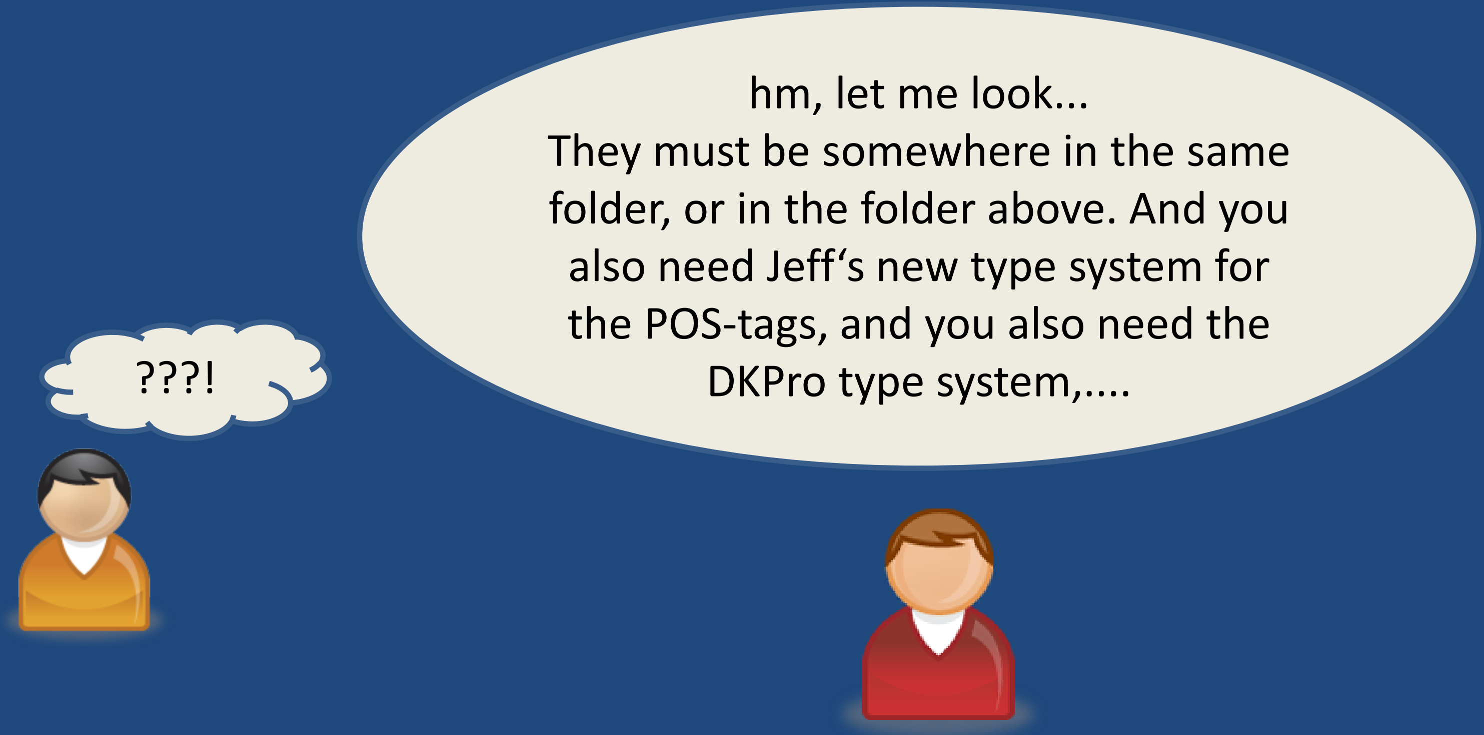
Sure, look here:
G:\MyFiles\docs\Peters\ner_2_31_15

I hope he has already shared that folder and I have the rights to access it...

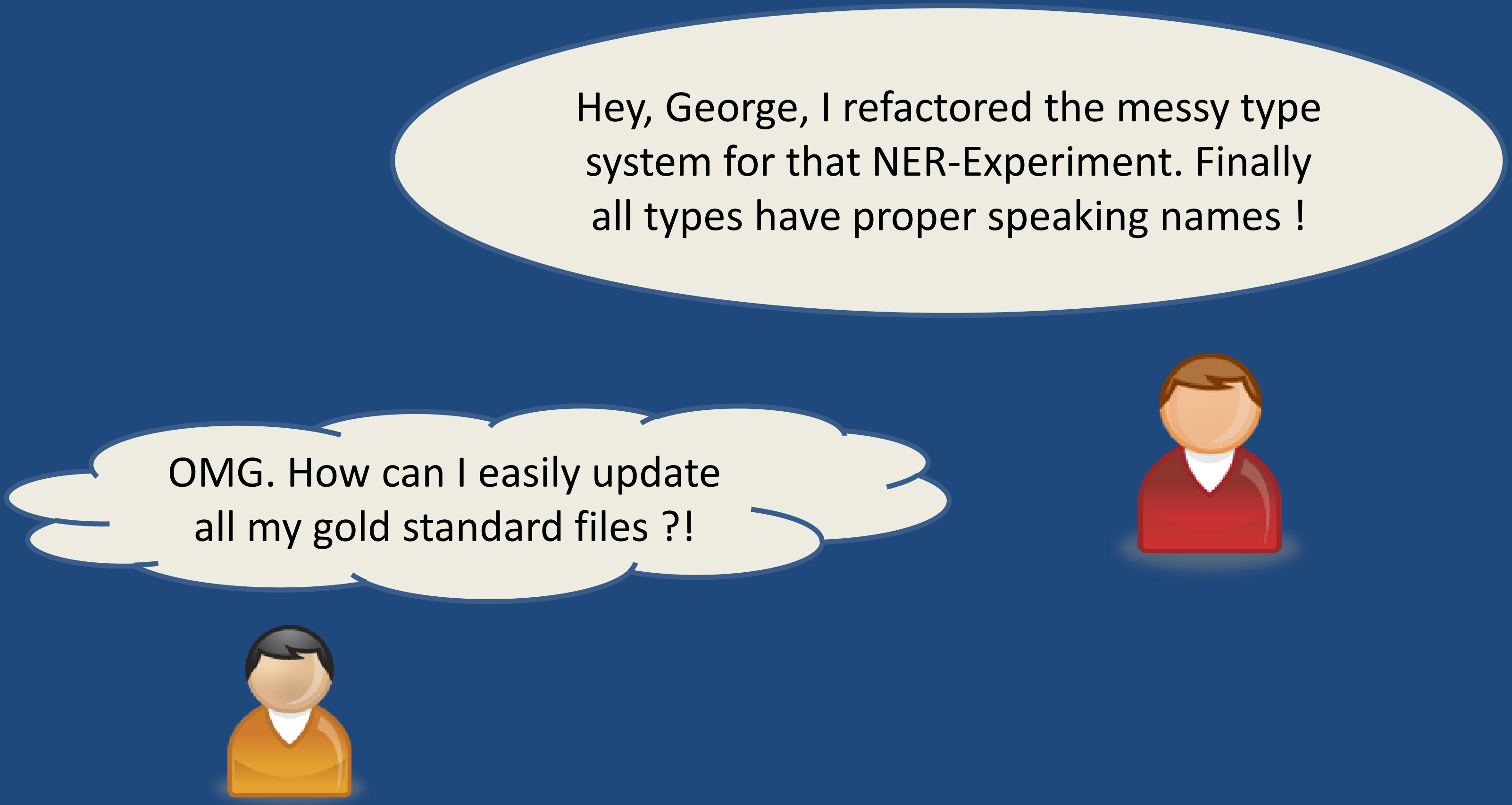
## 1. Problem: Organization of CASes and type systems



## 1. Problem: Organization of CASes and type systems


hm, let me look...
They must be somewhere in the same folder, or in the folder above. And you also need Jeff‘s new type system for the POS-tags, and you also need the DKPro type system,....
???!

## 2. Problem: Refactoring



Hey, George, I refactored the messy type system for that NER-Experiment. Finally all types have proper speaking names !

OMG. How can I easily update all my gold standard files ?!

## 3. Problem: Querying CAS collections for desired annotations/features

Hey George, quickly: How many noun phrases are in the parsed X corpus as an object for the verb „walk“ ?

Hm, ...Java program...iterate over all CASes...iterate over all noun phrase annotations...check for governing token „walk“...

## Possible solutions:

- Store CASes binary serialized in a data base

  But:

  - no index on the data
  - no search for proper type system for CASes

- Index engines on data (like Lucas, Fangorn, Tgrep)

  But:

  - no search for proper type system for CASes
  - some indices only designed for specific structures (e.g. Fangorn for parse tree banks)

- Query tools like RUTA-query-view

  But:

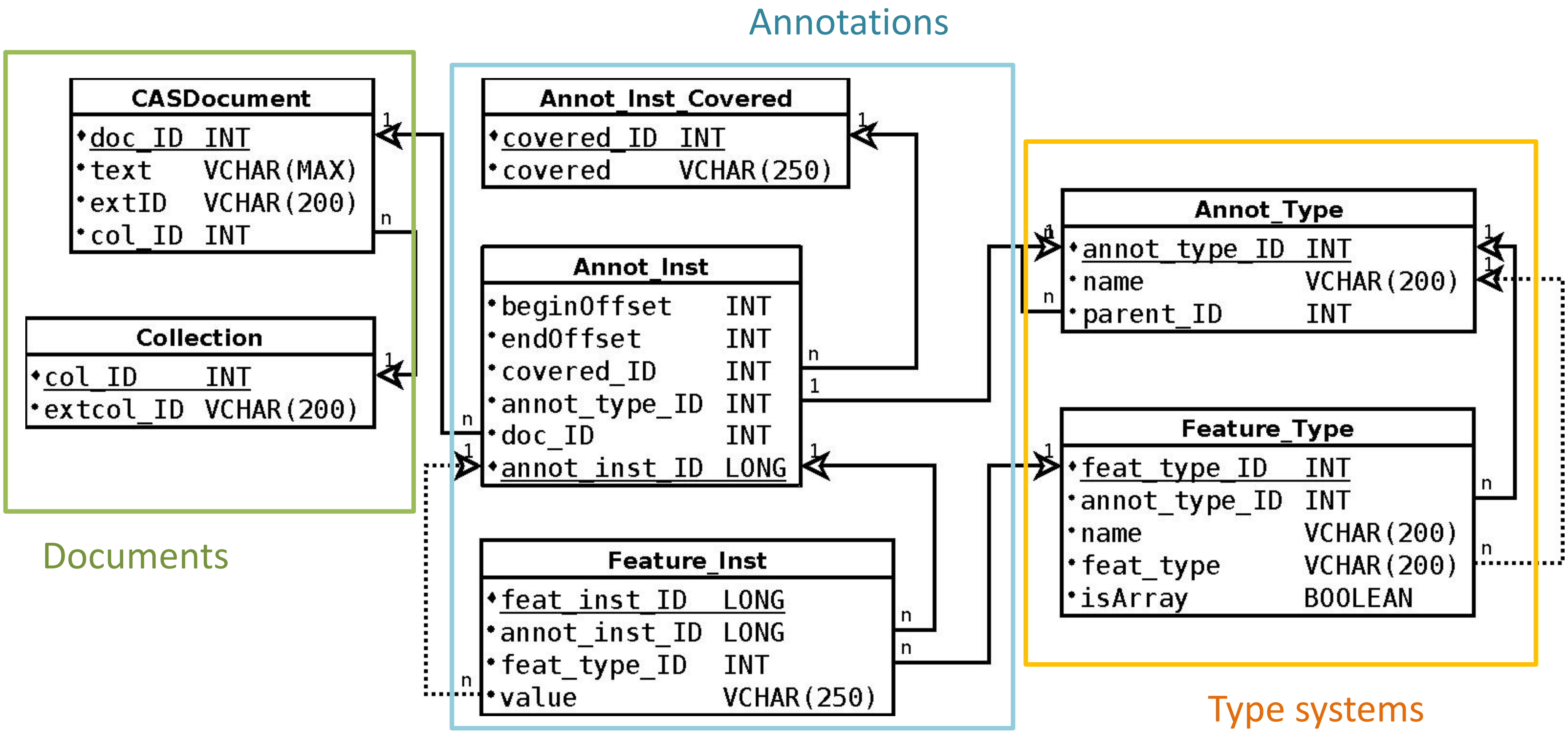
  - no index => slow search speed

Solution:

A storage possibility is needed which provides
- indices on the data
- combined storage of CASes and type systems
- easy access possibilities
- easy refactoring possibilities

=> Relational data base

# Solution: Storing CASes in a database



Annotations

Documents

Type systems

| CASDocument | |
|---|---|
| doc_ID | INT |
| text | VCHAR(MAX) |
| extID | VCHAR(200) |
| col_ID | INT |

| Collection | |
|---|---|
| col_ID | INT |
| extcol_ID | VCHAR(200) |

| Annot_Inst_Covered | |
|---|---|
| covered_ID | INT |
| covered | VCHAR(250) |

| Annot_Inst | |
|---|---|
| beginOffset | INT |
| endOffset | INT |
| covered_ID | INT |
| annot_type_ID | INT |
| doc_ID | INT |
| annot_inst_ID | LONG |

| Feature_Inst | |
|---|---|
| feat_inst_ID | LONG |
| annot_inst_ID | LONG |
| feat_type_ID | INT |
| value | VCHAR(250) |

| Annot_Type | |
|---|---|
| annot_type_ID | INT |
| name | VCHAR(200) |
| parent_ID | INT |

| Feature_Type | |
|---|---|
| feat_type_ID | INT |
| annot_type_ID | INT |
| name | VCHAR(200) |
| feat_type | VCHAR(200) |
| isArray | BOOLEAN |

## Solution: Storing CASes in a database

Java library which provides functionality for:

- saving/loading CASes
- creating CASCollectionReader/Writer
- loading/saving only parts of a collection of CASes (only texts/only selected Types)
- quickly delete selected types in a collection of CASes
- loading the type system for an arbitrary collection of CASes

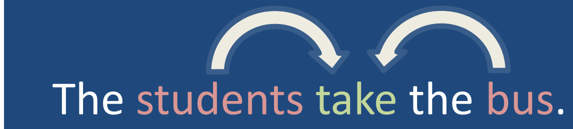
## Solution: Storing CASes in a database

Features available via direct SQL statements on the database:

- ➢ queries with database indices for
  - ▪ document-/annotation-texts (for pure textual queries)
  - ▪ existence/counts of specific annotations/features
  - ▪ structures of interlinked annotations/features (e.g. querying parse trees)
- ➢ renaming/refactoring of type names (simply rename the type in a type table, as all instances are references by internal IDs)

## query for structures of interlinked annotations/features:

*Which words are dependent of the word „take“in the parse trees of a corpus ?*



The students take the bus.

```
<typeDescription>
<name>Token</name>
<features>
<featureDescription>
<name>Governor</name>
<rangeTypeName>Token
</rangeTypeName>
</featureDescription>
</features>
</typeDescription>
```

***type system***

```
SELECT depText.covered FROM
  annot_inst govToken, annot_inst_covered govText,
  annot_inst depToken, annot_inst_covered depText,
  feat_inst, feat_type WHERE
feat_type.name = 'Governor' AND
govText.covered = 'take' AND
  depText.covered_ID = depToken.covered_ID AND
  feat_inst.annot_inst_ID = depToken.annot_inst_ID AND
  feat_inst.feat_type_ID = feat_type.feat_type_ID AND
  feat_inst.value = govToken.annot_inst_ID AND
  govText.covered_ID = govToken.covered_ID
```

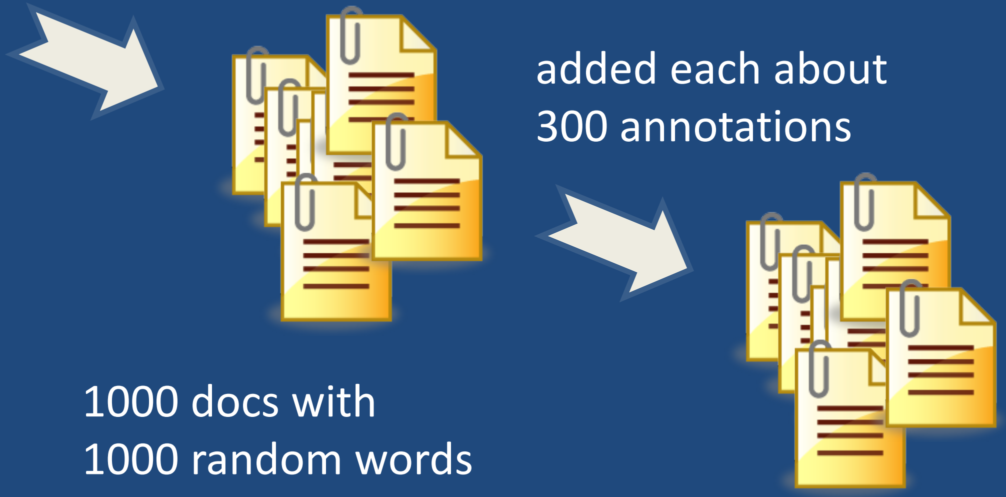
***SQL query for governed tokens***

# Evaluation on randomly created corpus

Annotation: „TestAnnot1“
  Feature1: String
  Feature2: Long

Dictionary of
1000 random
8-char words

- aakbvagj
- pcfqmhyk
- cspnjaaw
- ...

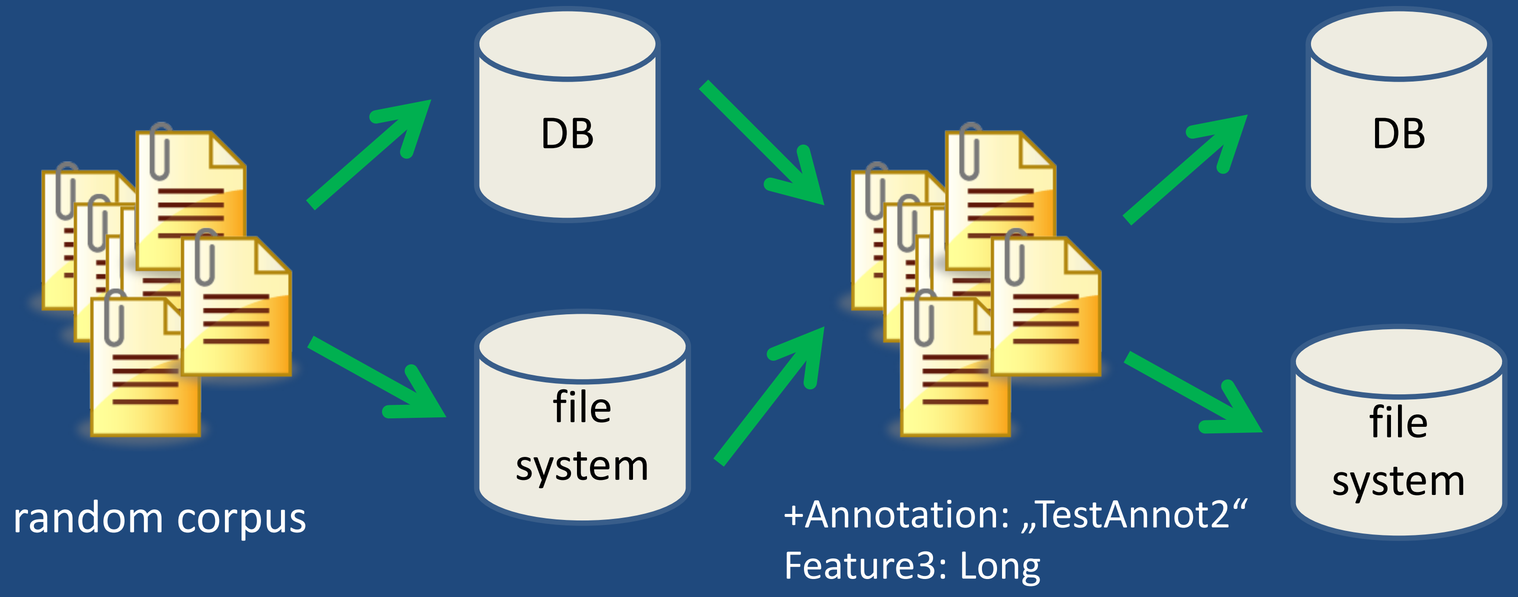


1000 docs with
1000 random words
from dictionary

added each about
300 annotations

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et ...

# Evaluation: experiment 1



DB

file system

random corpus

DB

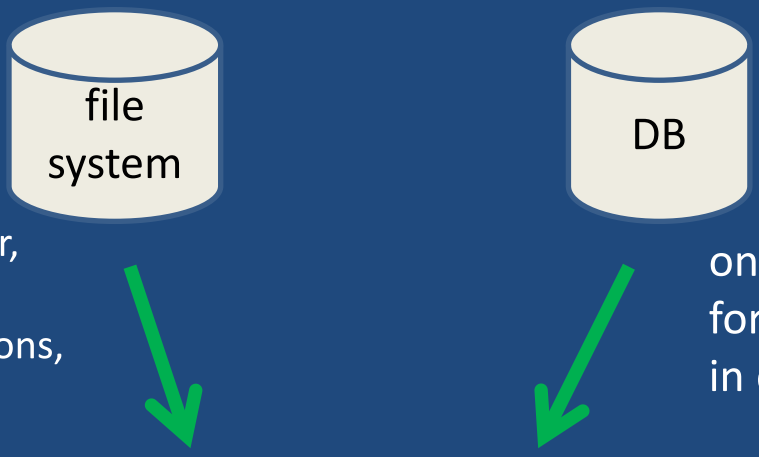
file system

+Annotation: „TestAnnot2“
Feature3: Long

| | save 1 (sec.) | load (sec.) | save 2 (sec.) |
|---|---|---|---|
| DB | 36.0 | 1.1 | 7.2 |
| file system | 2.6 | 1.1 | 2.7 |

# Evaluation: experiment 2

for all 1000 words X in the random corpus:
How many annotations of type „TestAnnot1“ with covered text X do exist ?



iterate over file folder,
deserialize CAS,
iterate over annotations,
accumulate counts

one SQL-query
for each word X
in corpus

List of counts for all words in corpus

| | query (sec.) |
|---|---|
| DB | 0.16 |
| file system | 7.0 |

Disadvantages/Future work:

- complicated SQL-queries get slow (e.g. parse tree queries with „tokens governing tokens governing tokens“)
- complicated SQL-queries get ugly (when the „easy“ SQL-queries do not yet look ugly enough...)

=> find a better way to improve query speed/look for complicated structure queries (perhaps integrating Fangorn)

- storing in database is slower than storing in file system, but depending on the scenario, storage performance is not that important

Advantages:

- useful query capabilities
- combination of annotations and type systems eleminate the search for proper type system files
- easy refactoring capabilities
- all UIMA data in one place

Give it a try:

http://code.google.com/p/uima-sql/

**Thank you for your attention.**