Vom Waldbau zum Wissensgraphen

Entwicklung einer RDF-Generierungspipeline mit SPARQL-Construct-Queries

Felix Vierthaler

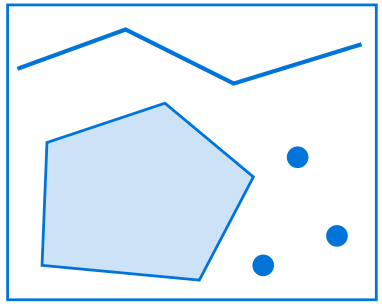
28.09.2026

Albert-Ludwigs-Universität Freiburg - Lehrstuhl für Algorithmen und Datenstrukturen

Wald im Klimawandel

- Wälder durch Klimawandel zunehmend gefährdet
- Natürliche Evolution des Waldes kommt nicht hinterher
- Menschen können manuell eingreifen, um Wald zu unterstützen
- Wie werden Entscheidungen getroffen?
  - Forstlicher Götterblick? Oder doch lieber Daten?
- Es existieren viele Daten über den Wald
  - Digitales Geländemodell, Waldhöhenstrukturkarte, Baumartenkarte, Bestandsdaten, Standortskartierung, Baumarteneignungskarten, Klimadaten, Wetterdaten …

→ Daten verteilt auf verschiedene Datenbanken, PDFs, Excel-Tabellen, Geodatabases, Shapefiles, TIFF-Dateien, Köpfe …

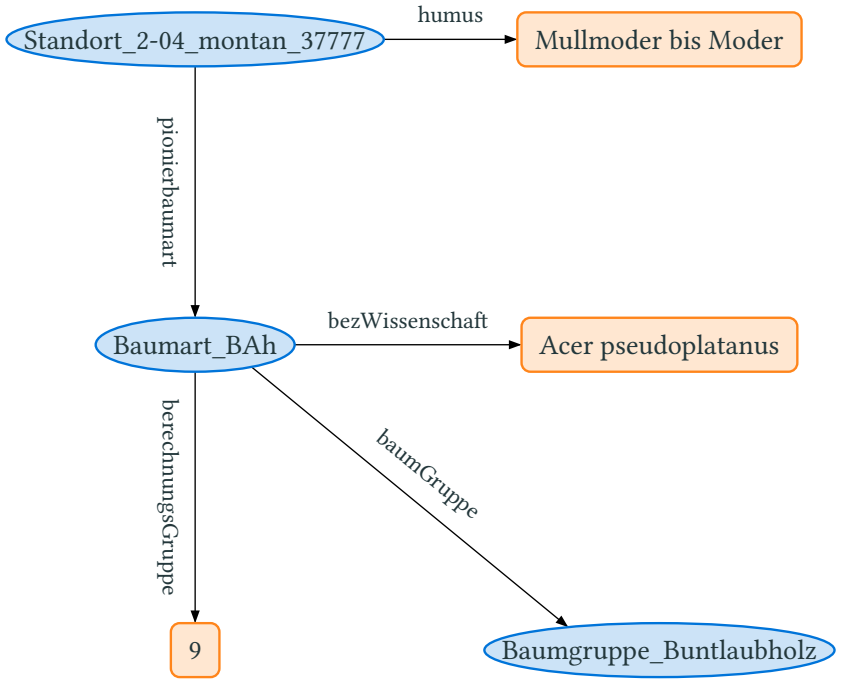


Vektordaten

10	25	40	55	60
20	45	70	60	45
35	65	85	50	30
30	50	55	35	15

Rasterdaten

Wissensgraphen: RDF (Resource Description Framework)



Alle Daten werden modelliert als Tripel:

Subjekt **Prädikat** **Objekt**

- Subjekte und Prädikate sind **IRIs**
  - Internationalized Resource Identifiers
  - "Wie eine URL" → global eindeutig
  - **Interoperabilität!**
- Objekte können IRIs oder **Literale** sein
  - Strings mit optionaler Datentyp-IRI
  - z.B. `"662669"^^xsd:int`

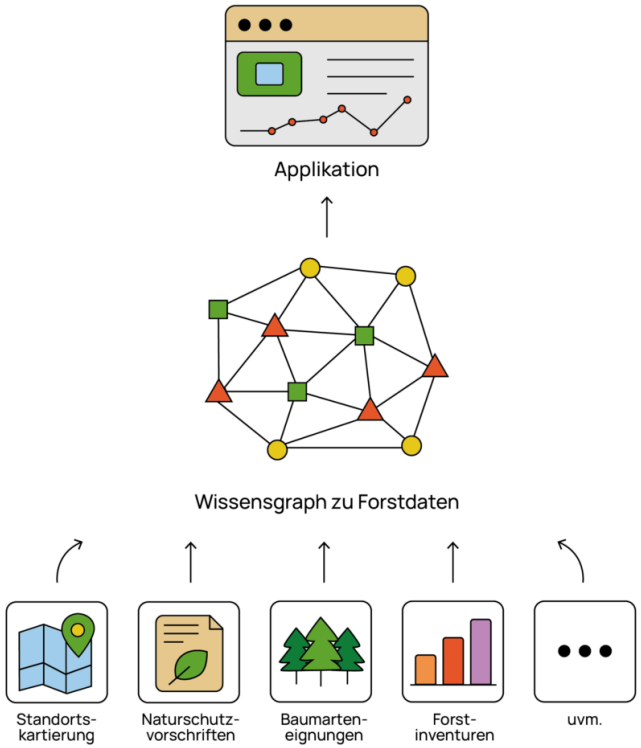
```
<wp:Standort_2-04_montan_37777> <wp:humus> "Mullmoder bis Moder" .
<wp:Standort_2-04_montan_37777> <wp:pionierbaumart> <wp:Baumart_BAh> .
<wp:Baumart_BAh> <wp:baumGruppe> <wp:Baumgruppe_Buntlaubholz> .
<wp:Baumart_BAh> <wp:bezWissenschaft> "Acer pseudoplatanus" .
<wp:Baumart_BAh> <wp:berechnungsGruppe> "9"^^xsd:int .
```

Anfragen an Wissensgraphen

- **SPARQL:** Variante der Datenbankabfragesprache SQL für RDF-Wissensgraphen
  - erlaubt komplexe Anfragen mit präziser Semantik und liefert strukturierte Ergebnisse aus Wissensgraphen zurück
- **Construct-Query:** Eine Anfrage, die als Ergebnis wieder RDF zurückgibt
- **QLever:** SPARQL-Engine des Lehrstuhls Algorithmen und Datenstrukturen
- **QLever-UI:** Moderne Web-Oberfläche zum Schreiben und Ausführen von SPARQL-Queries

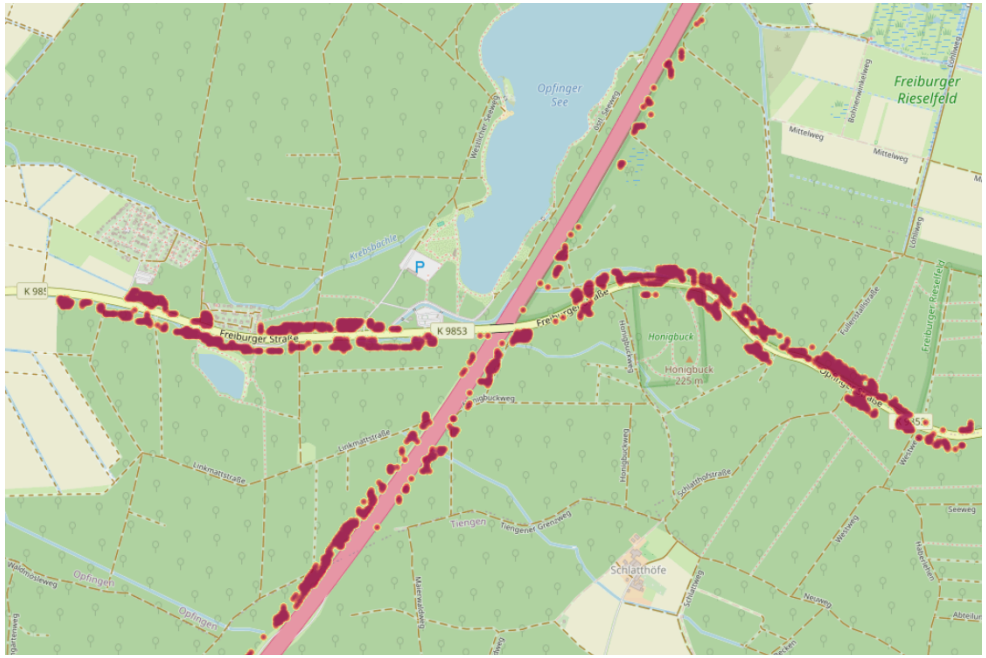
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX ex: <https://example.org/>
CONSTRUCT {
  ?stadt ex:einwohner ?einwohner .
  ?stadt ex:wkt ?wkt .
} WHERE {
  ?stadt rdf:type ex:Stadt .
  ?stadt ex:befindetSichIn ex:Badenwürttemberg .
  ?stadt ex:einwohner ?einwohner .
  ?stadt geo:hasGeometry/geo:asWKT ?wkt .
}
```

Forstgraph

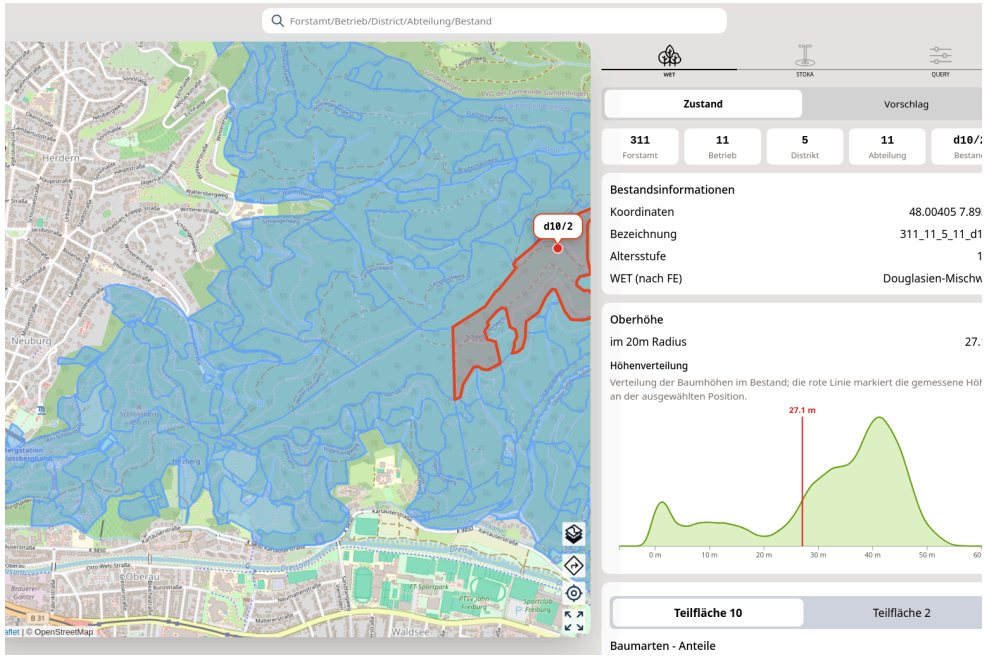


- Im Rahmen des **WWW-4.0-Projektes** (Wissen über den Wald im Wandel 4.0)
  - Zusammenschluss aus Soziologie, Informatik und Fachpersonen im Thema Forst
- **Ziel:** Wissensgraph zum Thema Forst in Baden-Württemberg aus bestehenden Datensätzen entwickeln → **Forstgraph**
- **Meine Rolle:** "Prototyp" des Forstgraphen entwickeln

Demos



Potenziell gefährdete Eschen



Demo-App

Qonstructor RDF-Generierungs-Pipeline

Qonstructor:

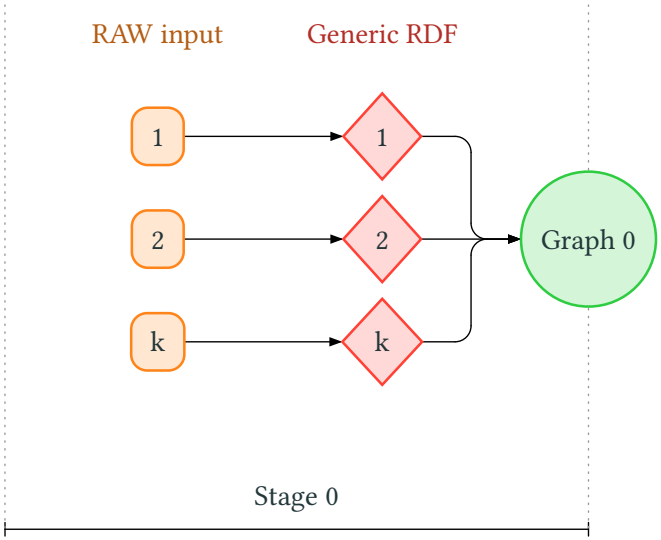
- Domänenunabhängige RDF-Generierungs-Pipeline
- Konfigurierbar mit YAML-Dateien
- Python-Programm, das verschiedene Prozesse orchestriert

Start von Qonstructor:

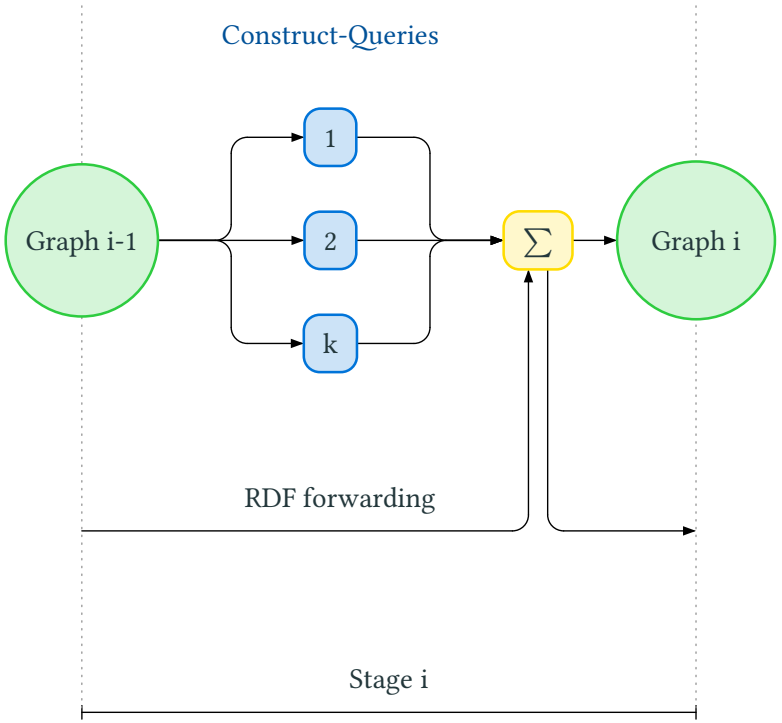
- Alle Konfigurationsdateien werden geparst und validiert
- Sequentielle Stages werden ausgeführt

Generic-Generation-Stage

- Source-Dateien werden mithilfe von **Generatoren** in generisches RDF umgewandelt
  - Ein Generator wandelt Dateien eines Dateityps in generisches RDF um
  - Ein Generator ist letztendlich ein beliebiges Shell-Kommando
  - Generatoren für CSV, JSON und TIFF-Dateien sind enthalten
  - Generatoren sind in Config-Dateien erweiterbar
- Aus dem resultierenden RDF wird ein **QLever-Index** gebaut und ein **QLever-Server** gestartet

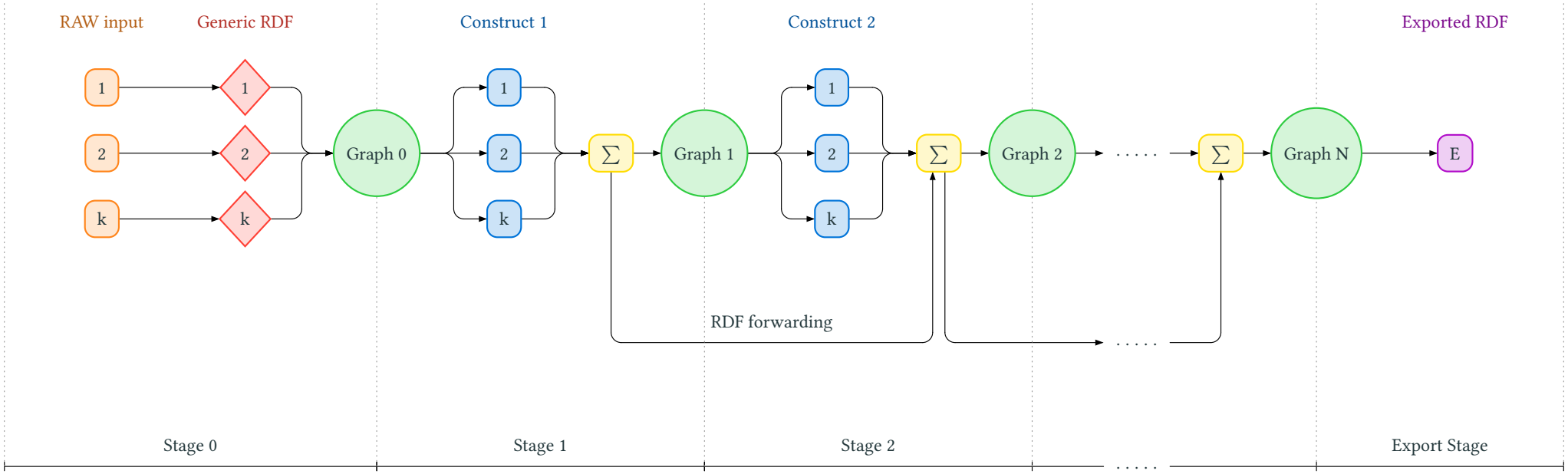


Construct-Stages 1-9



- Eine **Construct-Stage** macht Folgendes:
  - führt Construct-Queries auf dem vorherigen QLever-Server aus
    - Queries können in den Konfigurationsdateien definiert werden
  - Das resultierende RDF wird mit dem RDF aller vorherigen Construct-Stages kombiniert (RDF-Forwarding)
    - Damit wird ein QLever-Index gebaut und ein QLever-Server gestartet
- Das wird für alle Construct-Stages durchgeführt

Qonstructor-Pipeline-Architektur

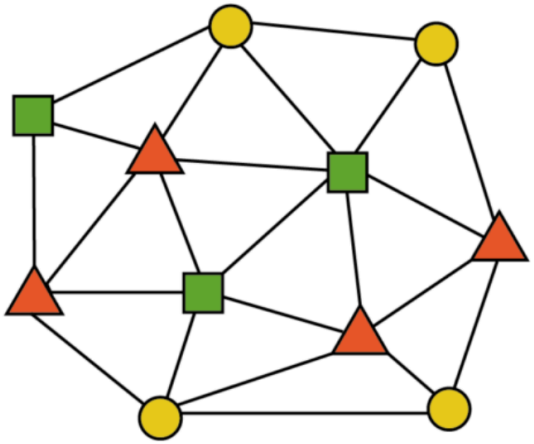


Warum?

- Warum **mehrere Construct-Stages**?
  - Es gibt Queries, die resultierende Triples einer vorherigen Query benötigen
- Warum keine **Update-Queries**?
  - Update-Queries können nicht parallel ausgeführt werden
  - Für Generierung von vielen Triples sind Construct-Queries und neuer Index-Bau schneller

Forstpipeline und Forstgraph

- **Vorarbeit**: Rohdateien aus diversen Formaten mit Skripten in die Formate CSV, JSON und TIFF umgewandelt
- **Forstpipeline**:
  - Nutzt Qonstructor, um Forstgraphen zu bauen
  - Besteht aus:
    - 34 Source-Dateien
    - 19 Stage-Konfigurationsdateien
    - 37 Construct-Queries
  - Nutzt 3 Construct-Stages:
    - Normalisierung - Linking - Inference
- **Forstgraph**:
  - 28 verschiedene Klassen
  - 121 verschiedene Prädikate
  - 117.075.599 Triples



Ontologische Grundlagen:

- Alle Objekte haben eine Klassenzugehörigkeit mit **`rdf:type`**
- Menschlich lesbare Bezeichnungen mit **`rdfs:label`**
- WKT-Geometrien mit **`geo:hasGeometry/geo:asWKT`**

IRI basics:

- Forstgraph nutzt das Präfix `wp:` (https://wald.cs.uni-freiburg.de/ontology#)
- **Klassen** haben IRI **`wp:Klassenname`** (z.B. `wp:Baumart`)
- **Entitäten** einer Klasse haben IRI **`wp:Klassenname_Objekt-Identifier`** (z.B. wp:Baumart_BAh)

IRI-Minting

- Wenn möglich **menschlich lesbare** IRIs (Minting via Concat)
  - z.B. **`wp:Standort_2-04_montan_37777`** oder **`wp:Baumart_BAh`**

```
IRI(CONCAT(STR(wp:Baumart_),?baumart))
```

- Sonst **deterministisch generierte** IRIs (Minting via SHA)
  - z.B. **`wp:Bestand_000053063288845a1`**

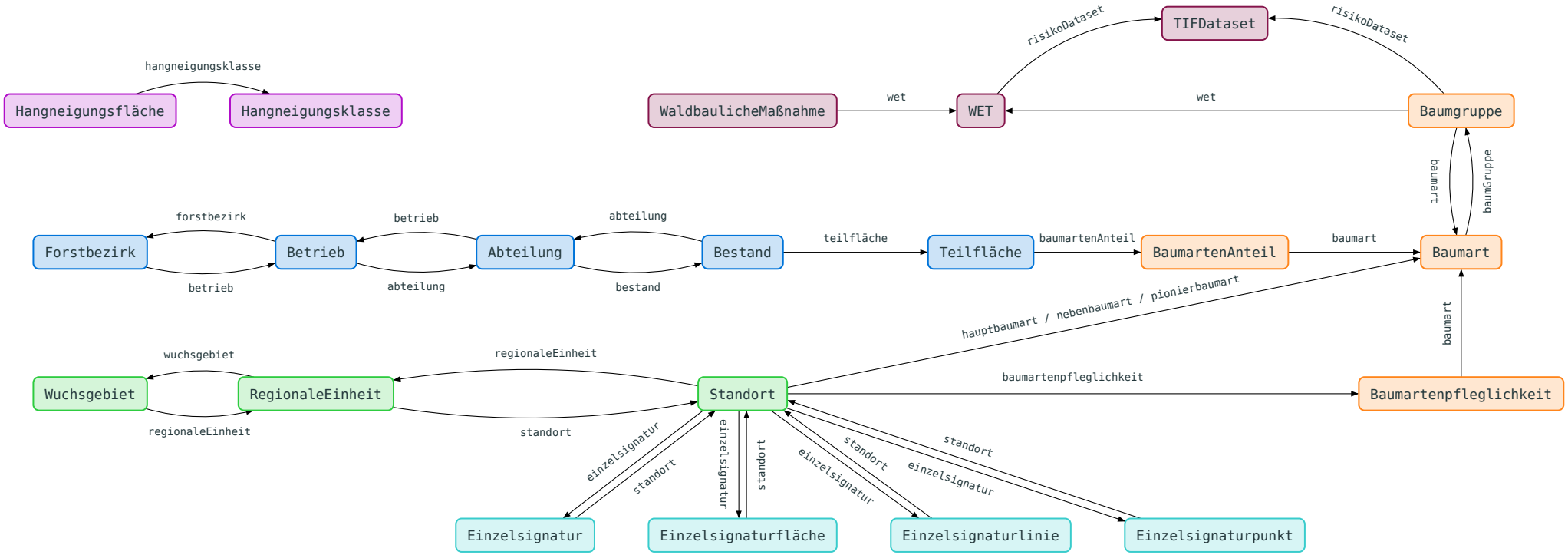
```
IRI(CONCAT(STR(wp:),"Vogelschutzgebiet_",(SUBSTR(SHA256(?WKT),1,16))))
```

Construct-Query-Beispiel (Stage 1 - Normalisierung)

```
PREFIX wp: <https://wald.cs.uni-freiburg.de/ontology#>
PREFIX dat: <data#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX gen: <generic#>

CONSTRUCT {
  ?iri rdf:type wp:RegionaleEinheit ;
    rdfs:label ?regionaleeinheit ;
    wp:nr ?nr .
  ?iri wp:wuchsgebiet ?wuchsgebiet_iri .  # link 1
  ?wuchsgebiet_iri wp:regionaleEinheit ?iri .  # link 2
} WHERE {
  gen:RegionaleGliederung gen:child ?row .
  ?row dat:Nr ?nr .
  ?row dat:RegionaleEinheit ?regionaleeinheit .
  ?row dat:WG_RE_F ?wg_re_f .
  BIND (IRI(CONCAT(STR(wp:), "RegionaleEinheit_", REPLACE(?nr, "/", "-"))) AS ?iri)  # primary iri
  BIND (IRI(CONCAT(STR(wp:), "Wuchsgebiet_", ?wg_re_f)) AS ?wuchsgebiet_iri)  # foreign iri
}
```

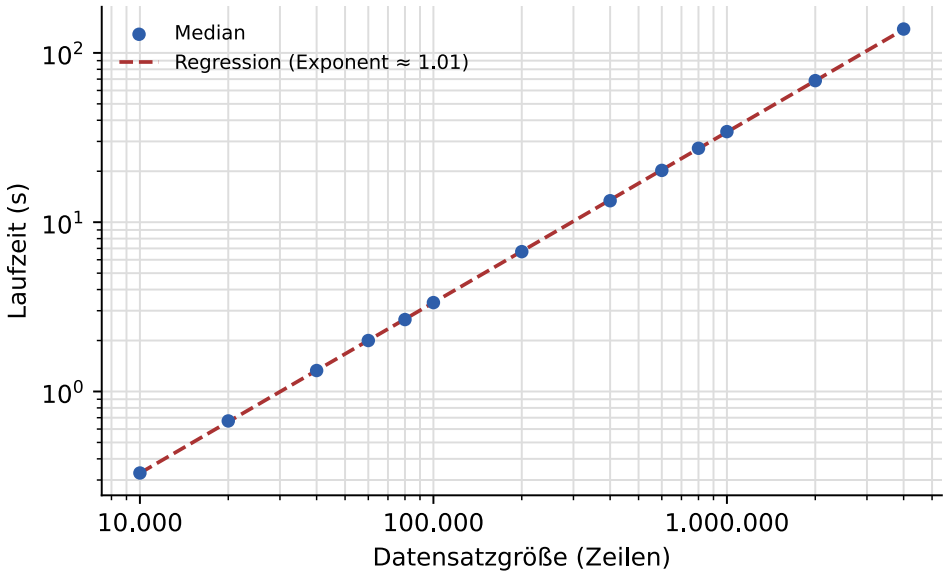
Ontologie im Überblick



Evaluation

- **Evaluationskriterien** der Forstpipeline:
  - **Qualität** des resultierenden Forstgraphen
    - Durch verschiedene Demo-Queries und Anwendungen gezeigt
  - **Gesamtlaufzeit** der Forstpipeline
    - Wichtig für stetige Weiterentwicklung
    - Ich habe die wichtigsten Aspekte untersucht

Laufzeit CSV2RDF

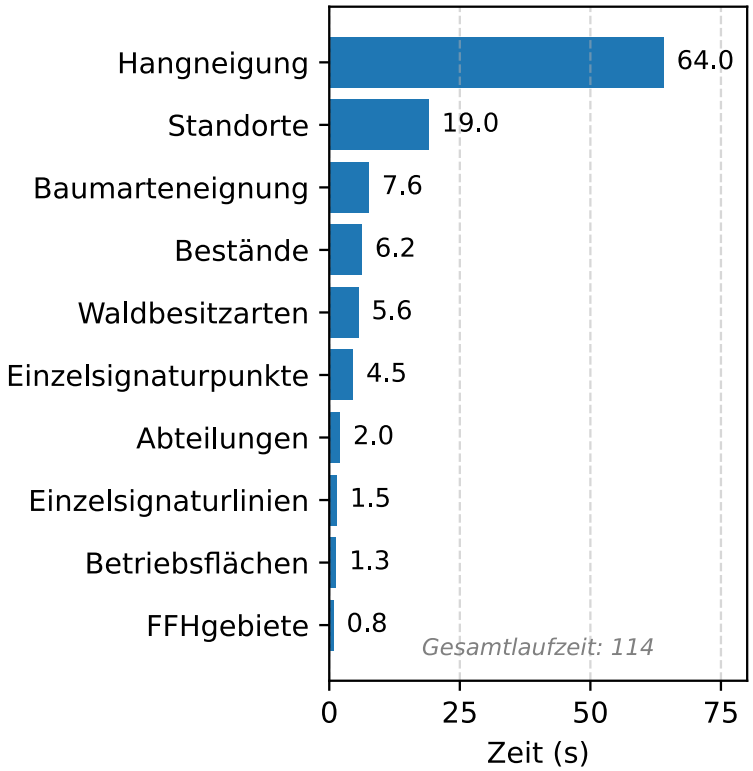


- Großteil der Sources sind **CSV-Dateien**
  - Werden von Generator-Programm CSV2RDF umgewandelt
- Laufzeit von **CSV2RDF**
  - Testdatensätze in verschiedenen Größen
  - Testdatensätze aus echten Source-Daten

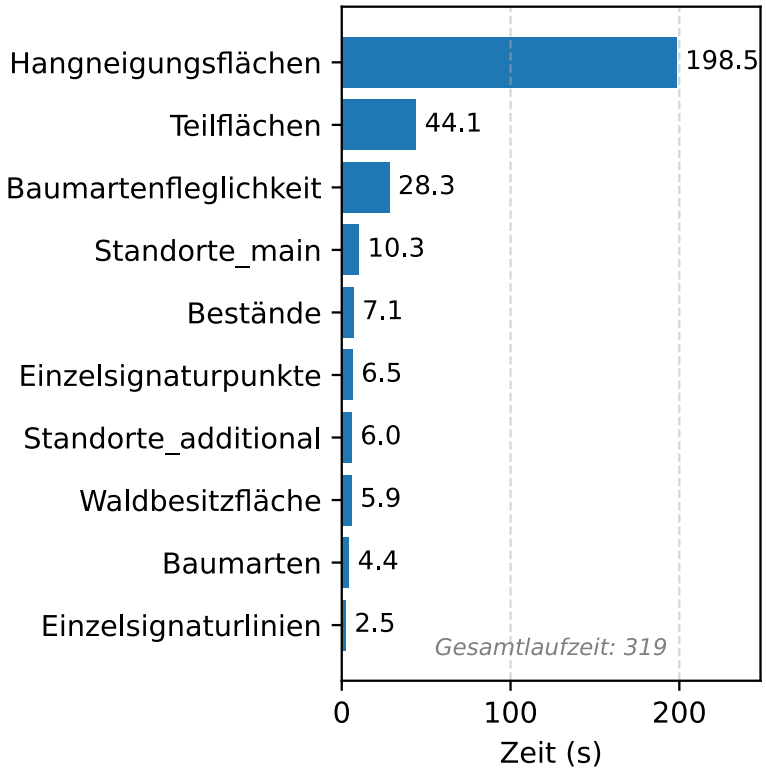
→ Laufzeit ist **annähernd linear**

Laufzeit pro Task (mit einem Worker)

- **Task**: Umwandlung einer Source-Datei / Ausführen einer Query

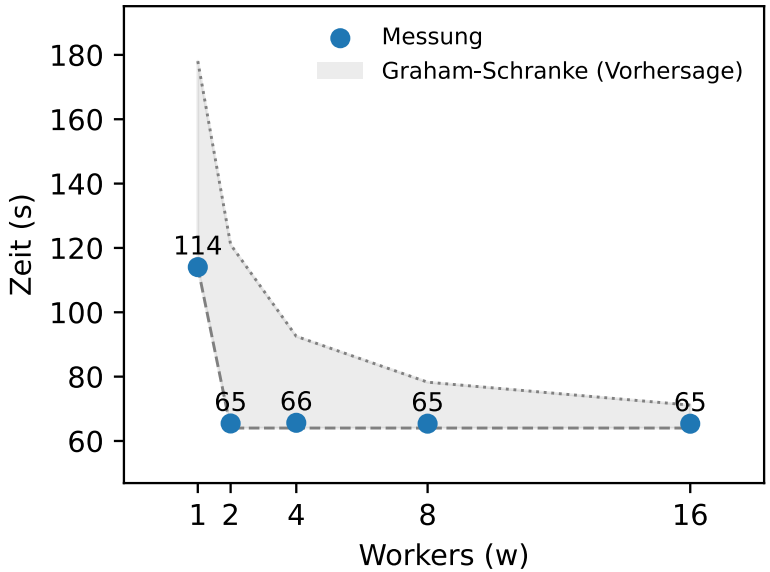


Laufzeit der Tasks in der Generic-Generation-Stage

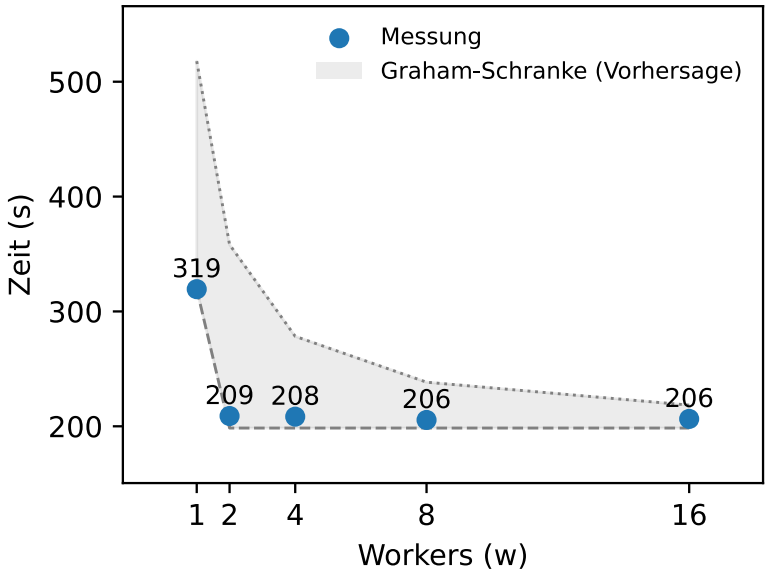


Laufzeit der Tasks in den Construct-Stages

Laufzeit der Stages mit verschiedenen Workern

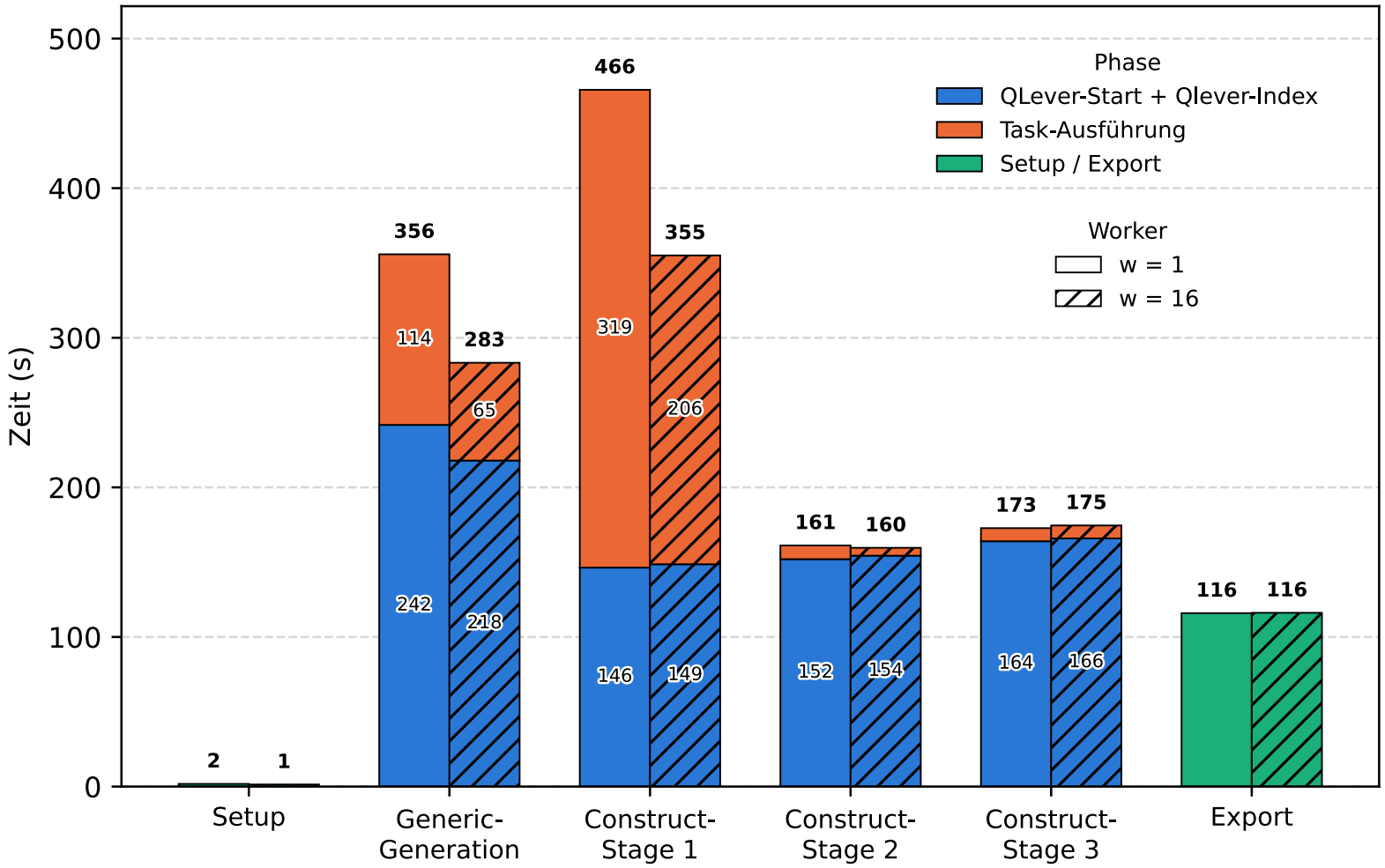


Laufzeit der Generic-Generation-Stage mit verschiedenen Workern

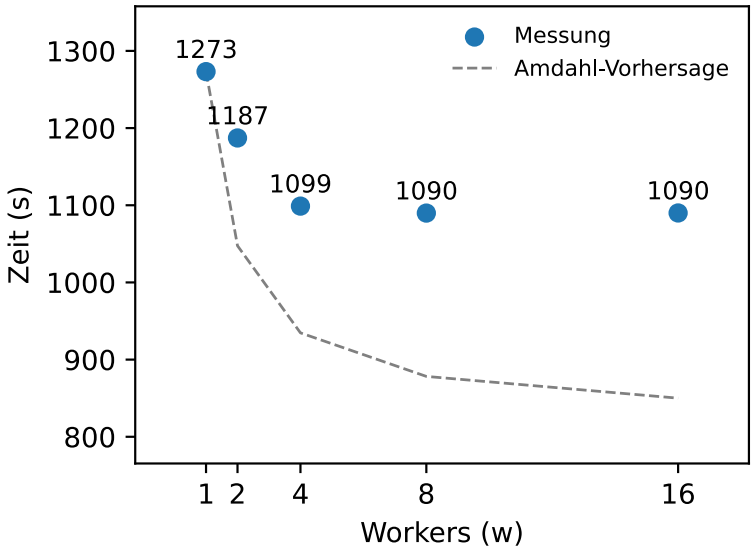


Laufzeit der ersten Construct-Stage mit verschiedenen Workern

Laufzeit aller einzelnen Stages für $W = 1$ und $W = 16$



Gesamtlaufzeit der Pipeline mit verschiedenen Workern



- **Verbesserungsmöglichkeiten**
  - Generatoren: Einzelne Source-Dateien parallel verarbeiten
  - Nur eine Construct-Stage, danach Update-Queries

Rückblick und Ausblick

- **Qonstructor** ist eine domänenunabhängige RDF-Generierungs-Pipeline, mit der sich komplexe RDF-Wissensgraphen modellieren lassen
- Der **Forstgraph** ist ein RDF-Wissensgraph zum Thema Forst in Baden-Württemberg, mit dem reale Use-Cases gelöst werden können
  - Kann "einfach" um mehr Daten erweitert werden
  - Erfolgreicher Prototyp
- Langfristige **Verbesserungsmöglichkeiten**
  - Datenvalidierung
  - Daten regelmäßig automatisch updaten
  - Daten-Lineage

Danke für die Aufmerksamkeit!

Externe Bildquellen

- https://uni-freiburg.de/www40/