Scaling Parallel Rule-based Reasoning

Martin Peters[1], Christopher Brink[1], Sabine Sachweh[1] and Albert Zündorf[2]

[1] University of Applied Sciences and Arts Dortmund, Germany
Smart Environments Engineering Lab

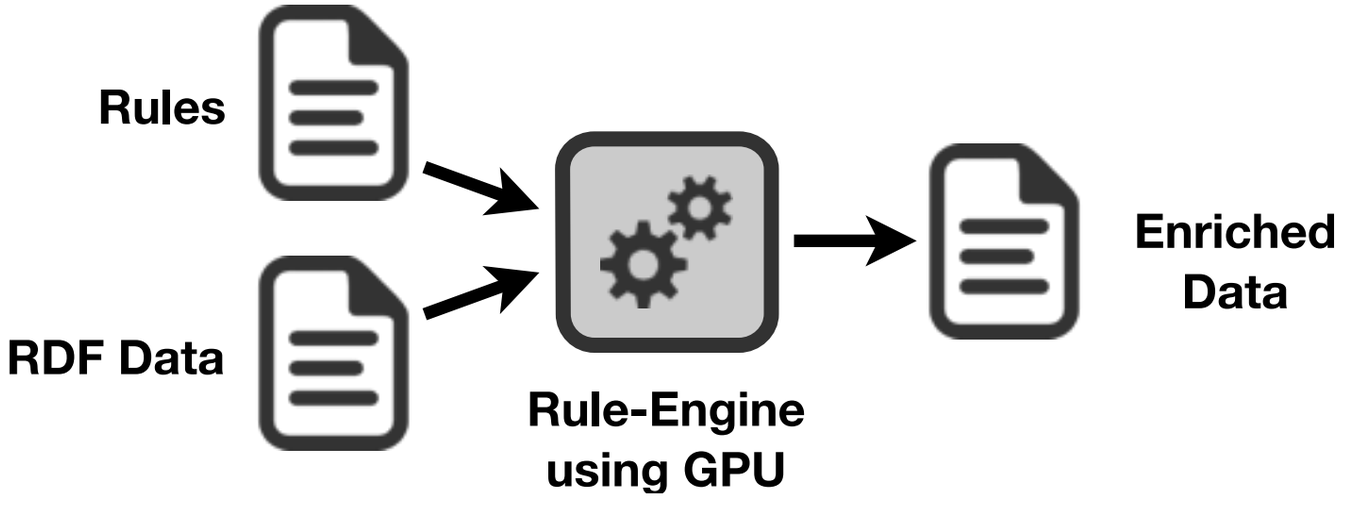
[2] University of Kassel, Germany,
Software Engineering Research Group

Motivation

- Reasoning is one key feature when using ontologies
- Reasoning means to create new knowledge by inferring facts that are implicitly given by the existing data
- The ontology language, that defines which triples should be inferred, can often be described in a rule-based way
  - ρdf (subset of RDFS)
  - RDFS
  - pD* (also known as OWL-Horst)
- Application specific rule-sets are also possible
- But: reasoning can still be a very time consuming task, depending on
  - the dataset
  - the used ontology language

Previous work

- **[PBSZ13]: Rule-based Reasoning on Massively Parallel Hardware**
  Martin Peters, Christopher Brink, Sabine Sachweh, Albert Zündorf
  9th International Workshop on Scalable Semantic Web Knowledge Base Systems @ ISWC 2013, October 2013, Sydney, Australia
  - a RETE-based forward chaining reasoner implementation that runs on a GPU



- allows to define the rules that shall be applied in a simple rule-file
- shows good performance for RDFS and OWL-Horst

Limitations

- limited size of processable input data
  - semantic data (dictionary encoded) of one RETE processing-step had to be loaded into the main memory of the GPU
  - total size of processable input data was limited by the size of the onboard memory of the used GPU

- only one GPU could be used
  - not scaleable in terms of using multiple GPUs

- for some rule-sets rule firing was one of the most time consuming tasks

How to overcome these limitations?

- Introducing workload partitioning:
  - computational load is partitioned into smaller chunks
  - each chunk can be processed independently
  - they can be distributed to multiple GPUs
  - these chunks can be sized with respect to a target device (GPU)

- Parallelizing the rule-firing
  - new triples are derived on the GPU
  - concept for reducing invalid triples (like duplicates) is introduced

Basic RETE algorithm (from [PBSZ13])

The RETE network

R1: (?x ?p ?y) → (?p rdf:type rdf:Property)

R2: (?x ?p ?y) (?p rdfs:domain ?c) → (?x rdf:type ?c)

The RETE network

R1: (?x ?p ?y) → (?p rdf:type rdf:Property)

R2: (?x ?p ?y) (?p rdfs:domain ?c) → (?x rdf:type ?c)

α1

The RETE network

R1: (?x ?p ?y) → (?p rdf:type rdf:Property)

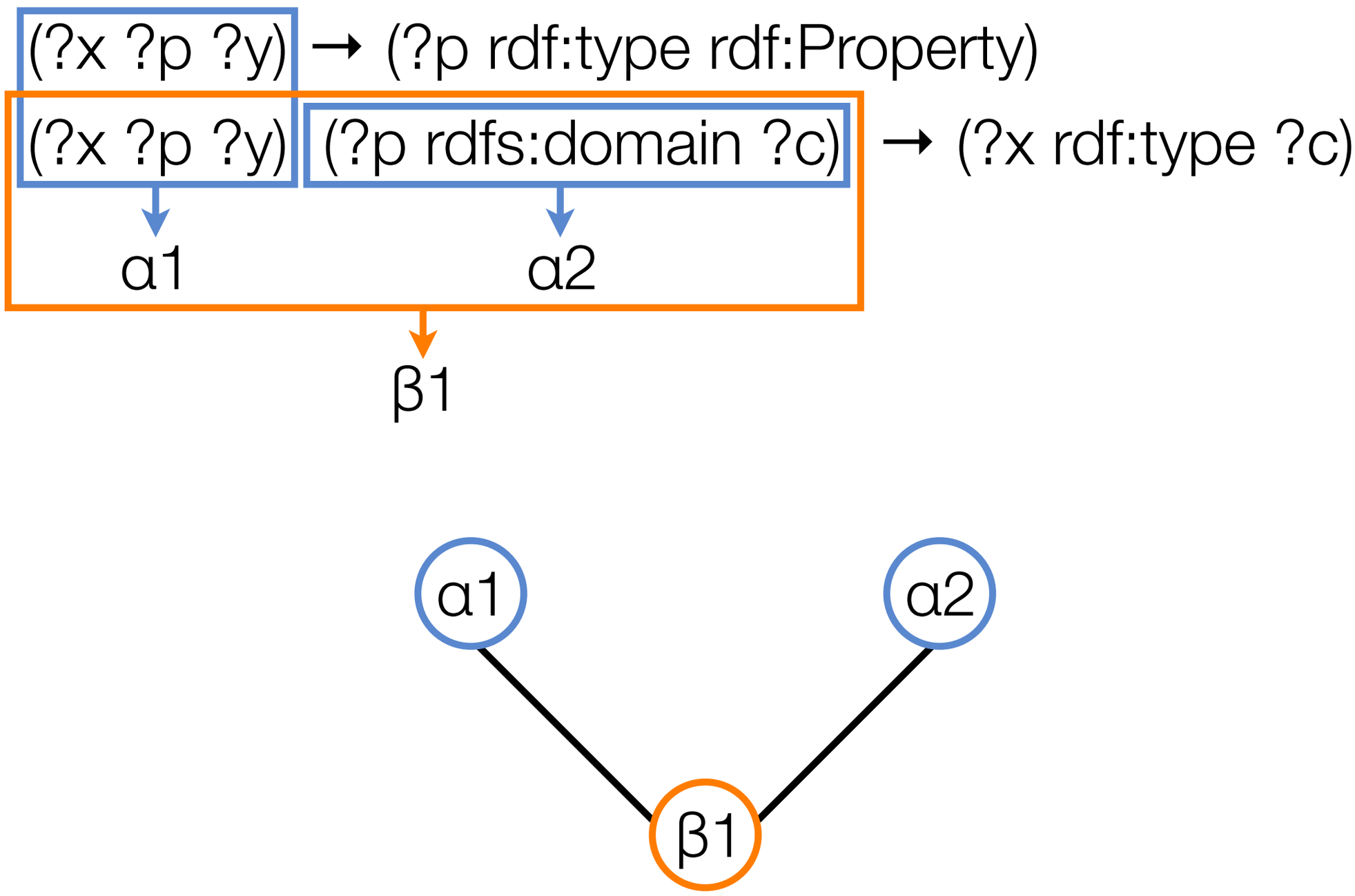
R2: (?x ?p ?y) (?p rdfs:domain ?c) → (?x rdf:type ?c)

α1 α2

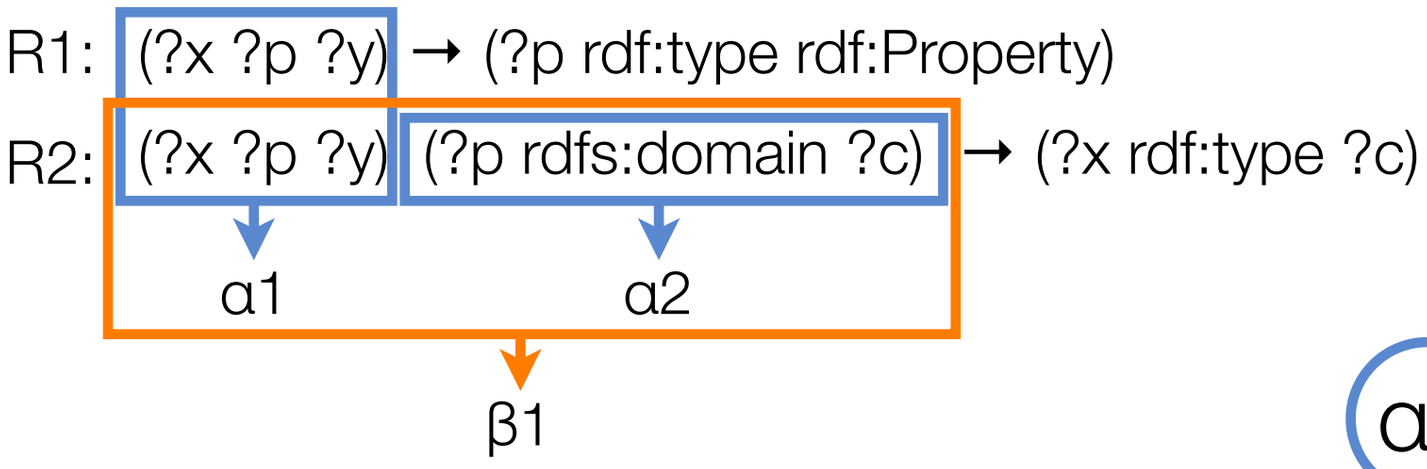
The RETE network

R1: (?x ?p ?y) → (?p rdf:type rdf:Property)

R2: (?x ?p ?y) (?p rdfs:domain ?c) → (?x rdf:type ?c)



alpha- and beta matching

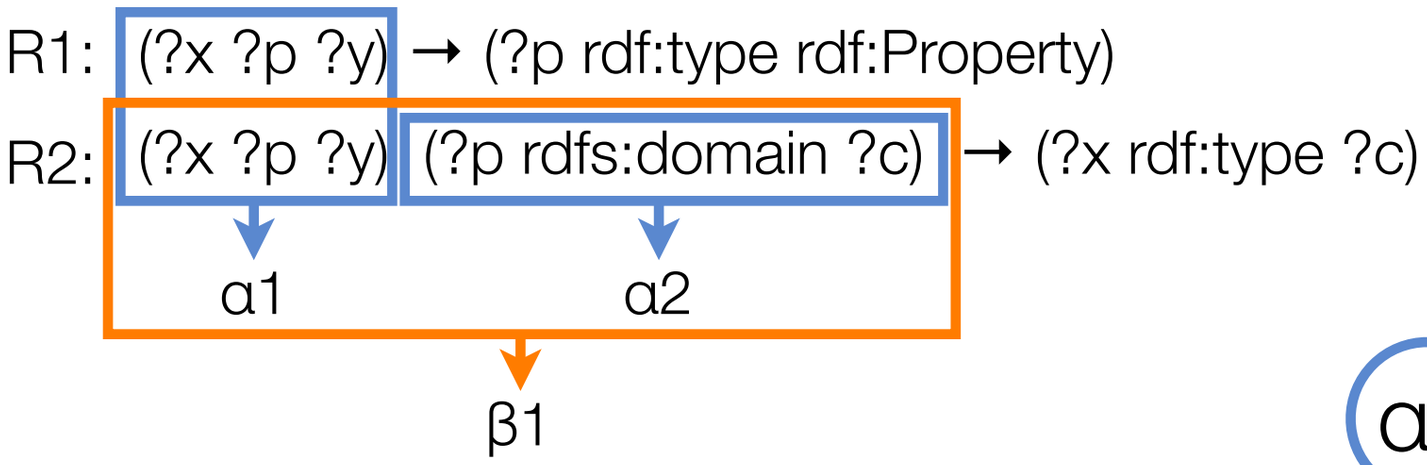


T1: [Bob uni:publishes Paper1]

T2: [Alice uni:publishes Paper2]

T3: [uni:publishes rdfs:domain Researcher]

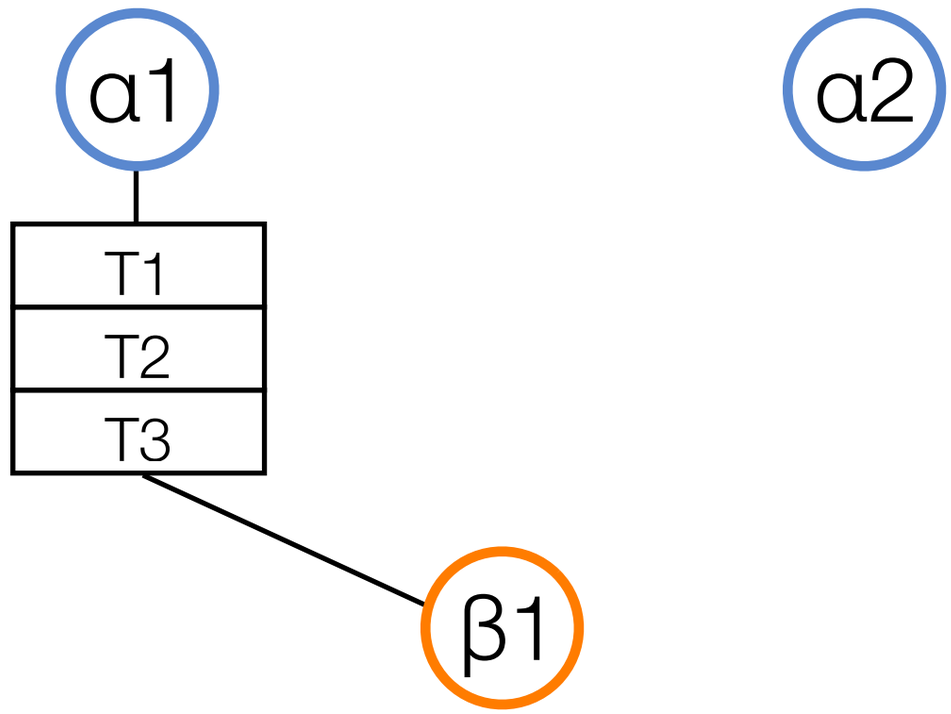
alpha- and beta matching



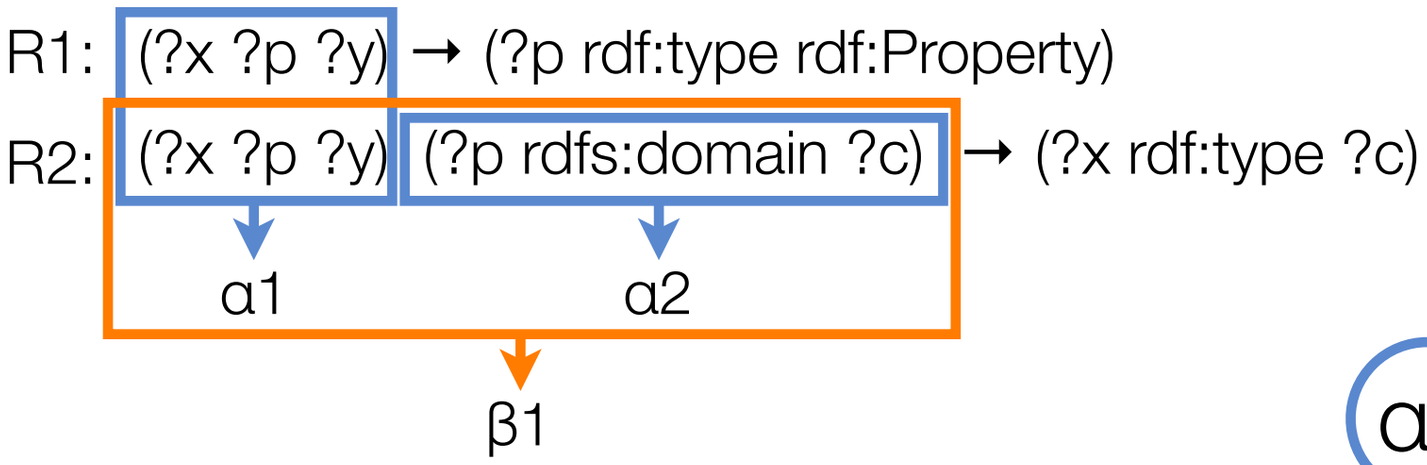
T1: [Bob uni:publishes Paper1]

T2: [Alice uni:publishes Paper2]

T3: [uni:publishes rdfs:domain Researcher]



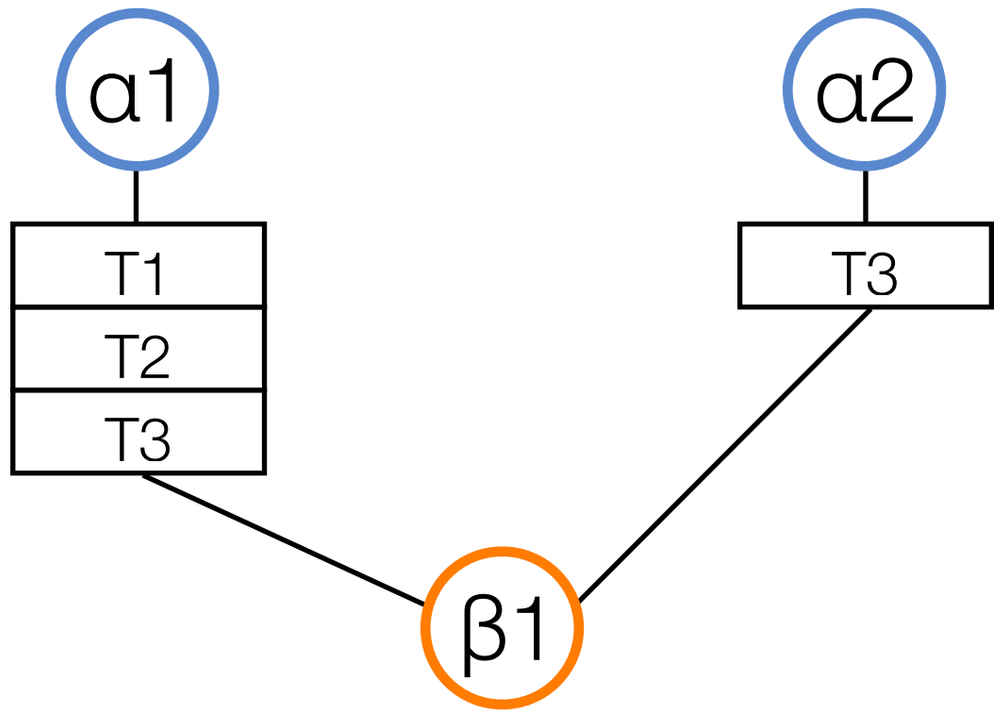
alpha- and beta matching



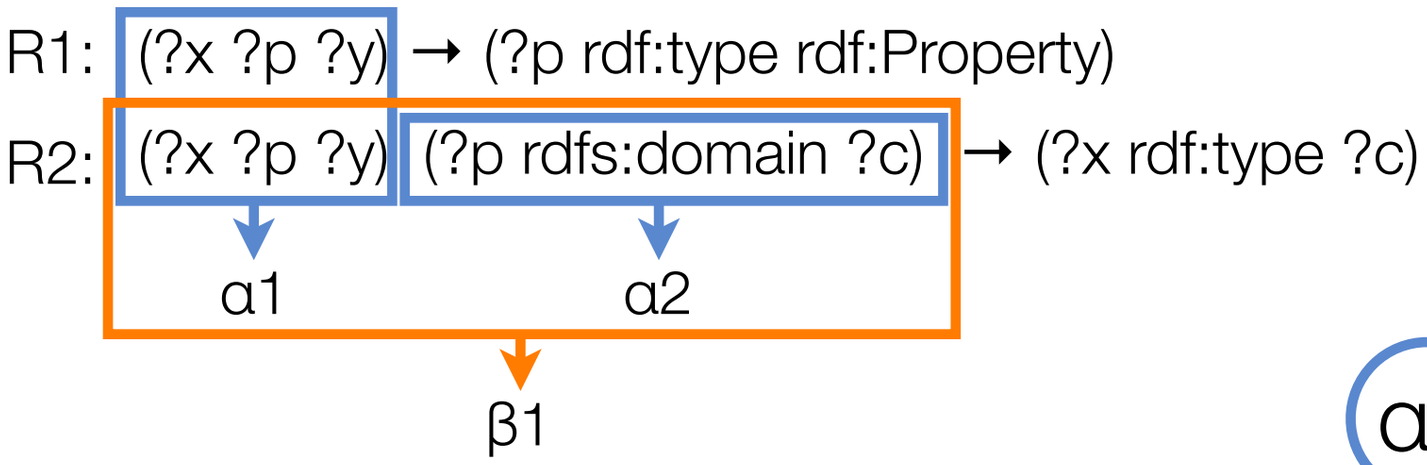
T1: [Bob uni:publishes Paper1]

T2: [Alice uni:publishes Paper2]

T3: [uni:publishes rdfs:domain Researcher]



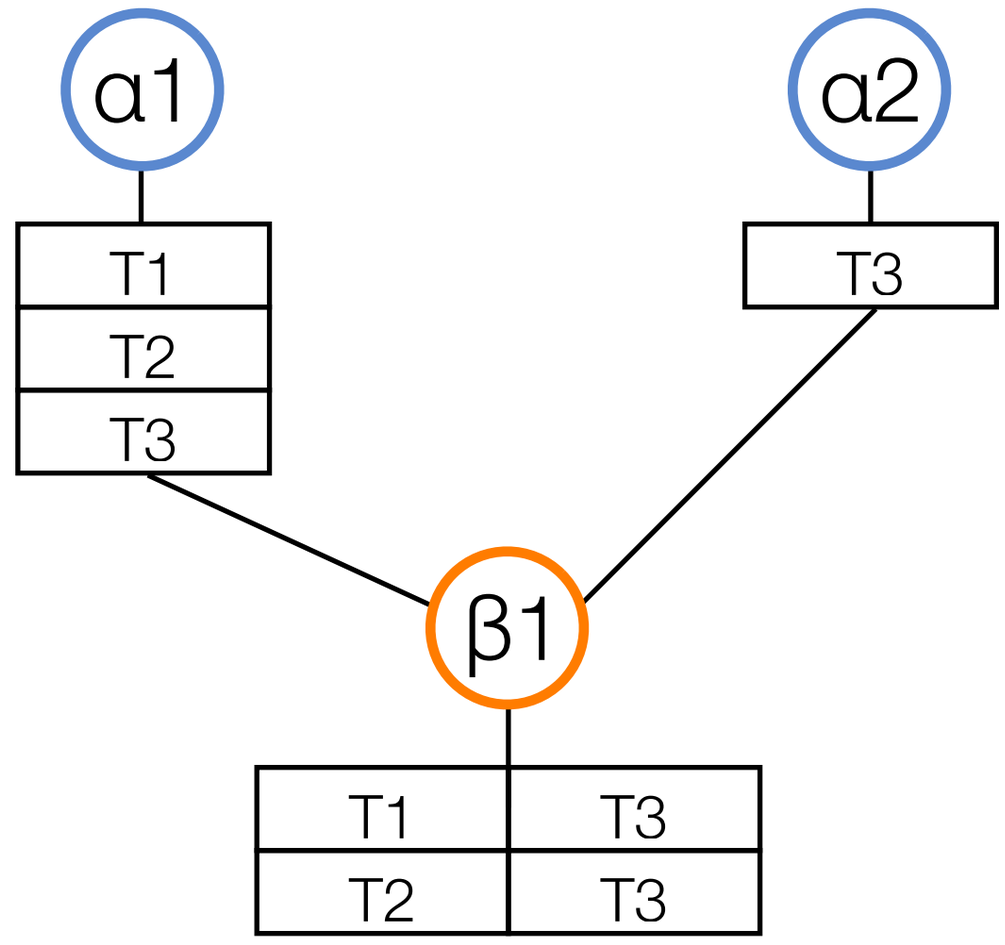
alpha- and beta matching



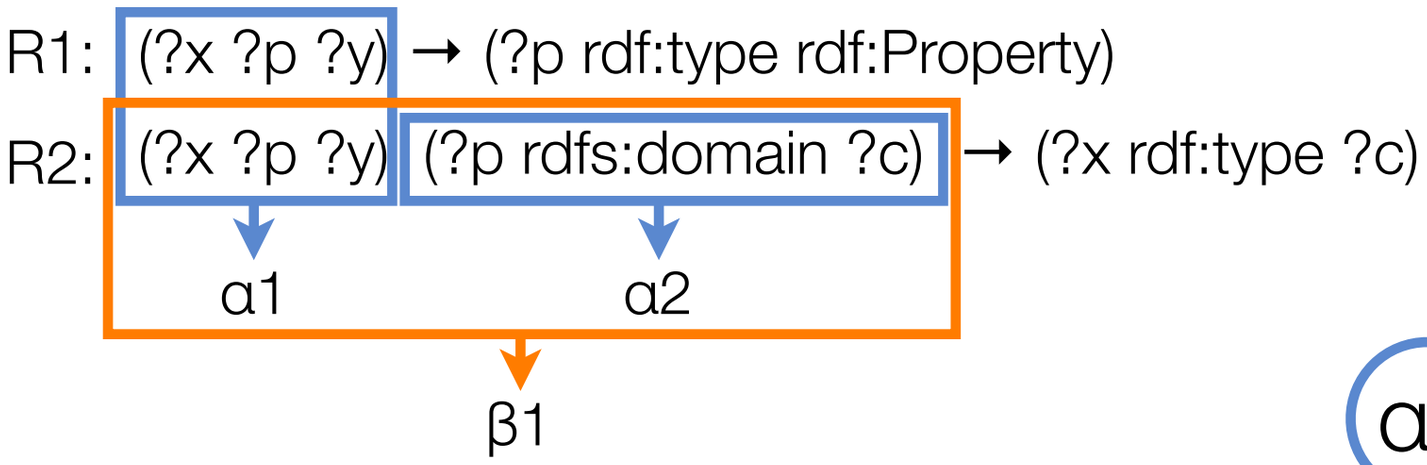
T1: [Bob uni:publishes Paper1]

T2: [Alice uni:publishes Paper2]

T3: [uni:publishes rdfs:domain Researcher]



alpha- and beta matching



T1: [Bob uni:publishes Paper1]

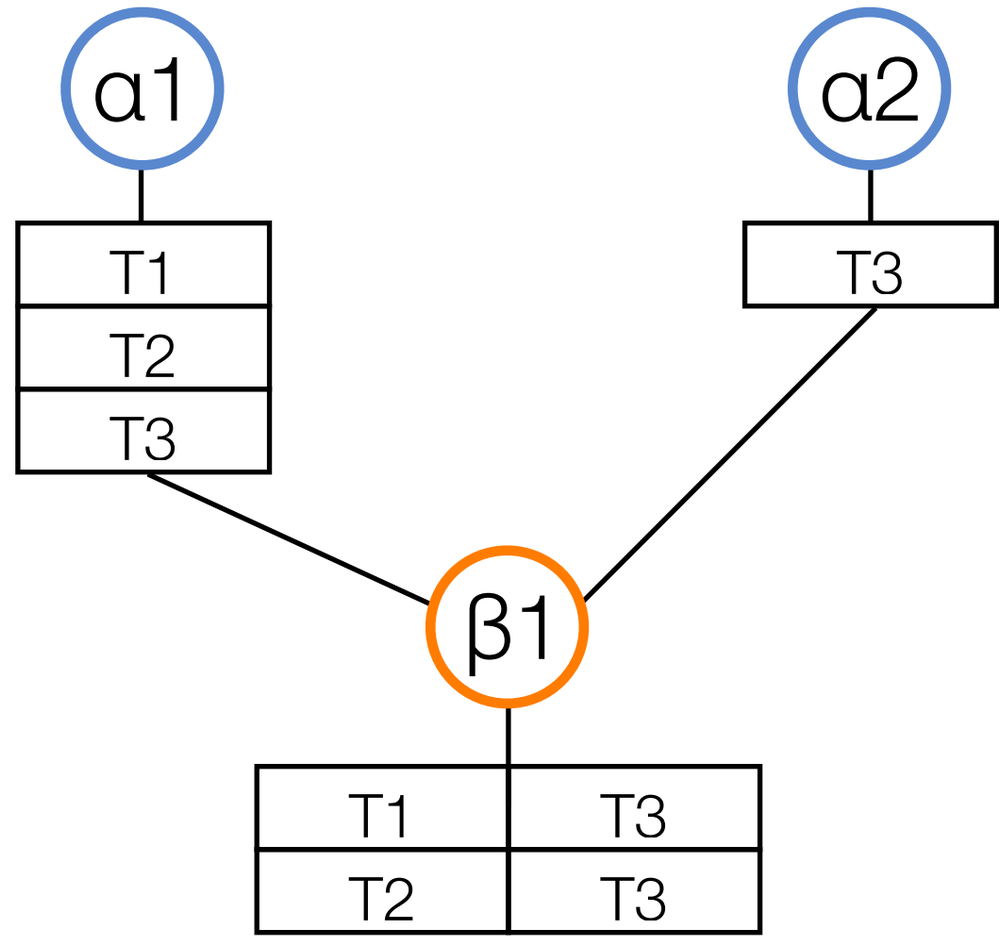
T2: [Alice uni:publishes Paper2]

T3: [uni:publishes rdfs:domain Researcher]

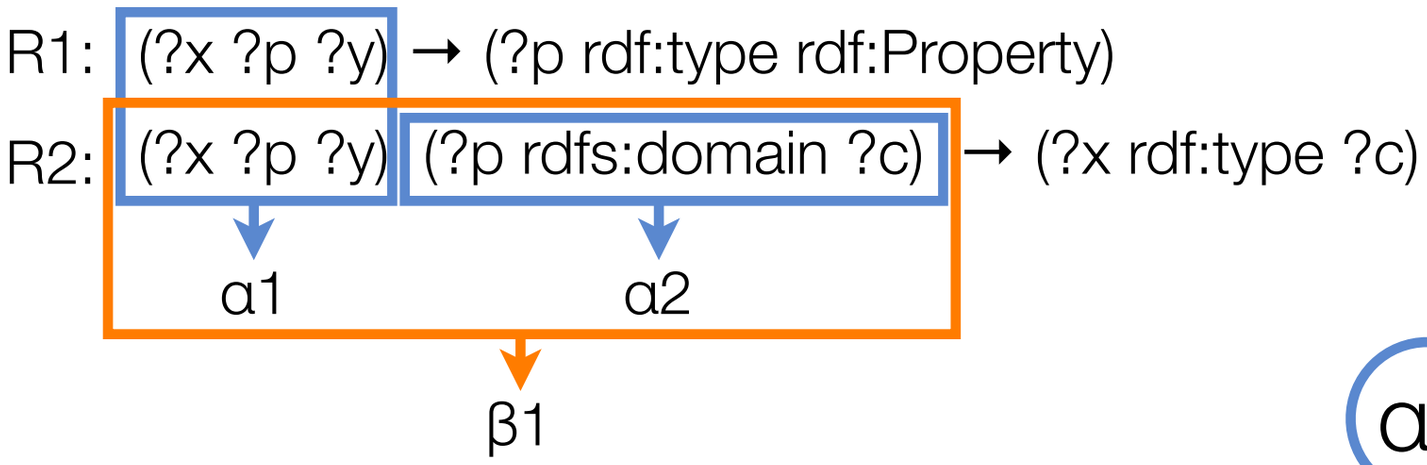
T4: [uni:publishes rdf:type rdf:Property]

[uni:publishes rdf:type rdf:Property]

T5: [rdfs:domain rdf:type rdf:Property]



alpha- and beta matching



T1: [Bob uni:publishes Paper1]

T2: [Alice uni:publishes Paper2]

T3: [uni:publishes rdfs:domain Researcher]

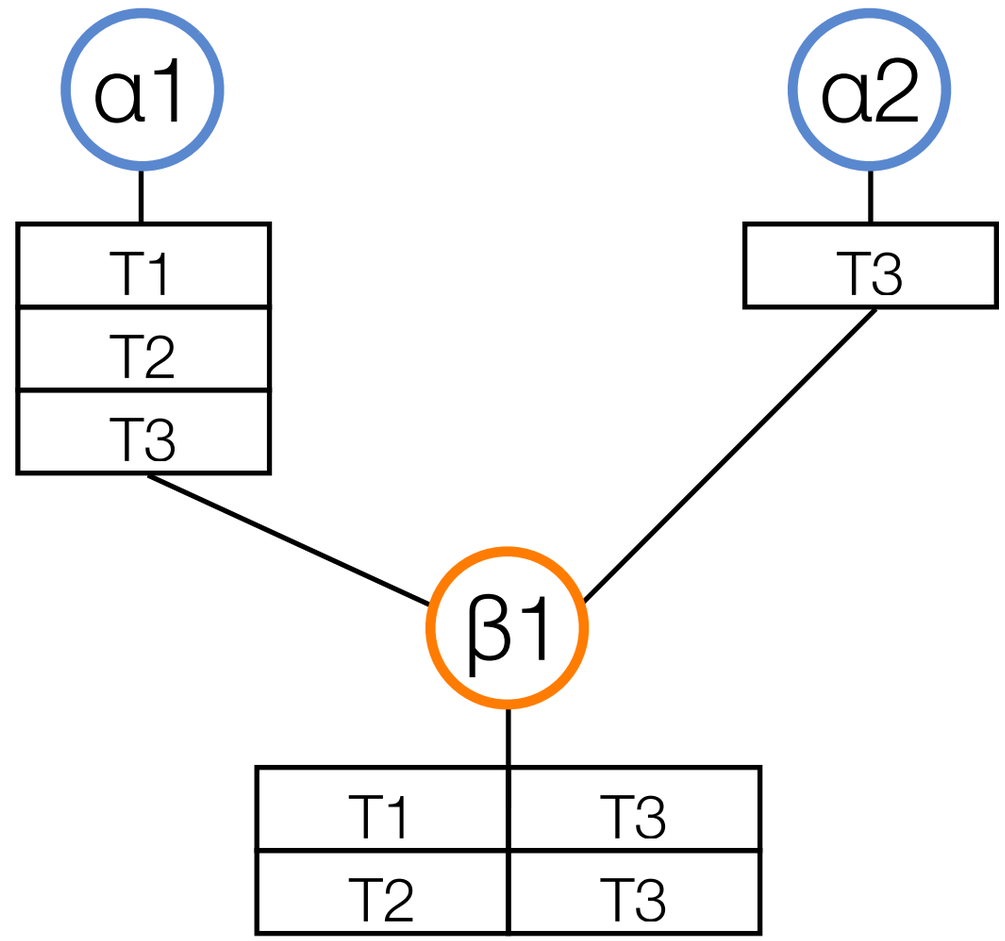
T4: [uni:publishes rdf:type rdf:Property]

[uni:publishes rdf:type rdf:Property]

T5: [rdfs:domain rdf:type rdf:Property]

T6: [Bob rdf:type Researcher]

T7: [Alice rdf:type Researcher]



Basic RETE algorithm (from [PBSZ13])

Parallelization: targeting massively parallel Hardware

Basic RETE algorithm (from [PBSZ13])

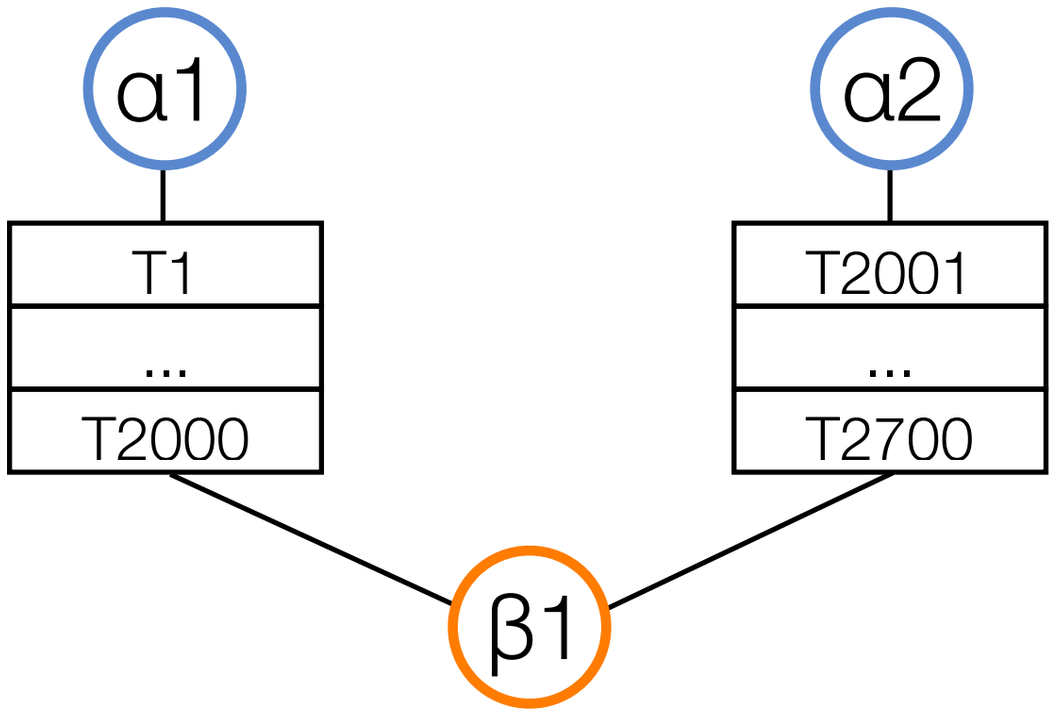
Parallelization: targeting massively parallel Hardware

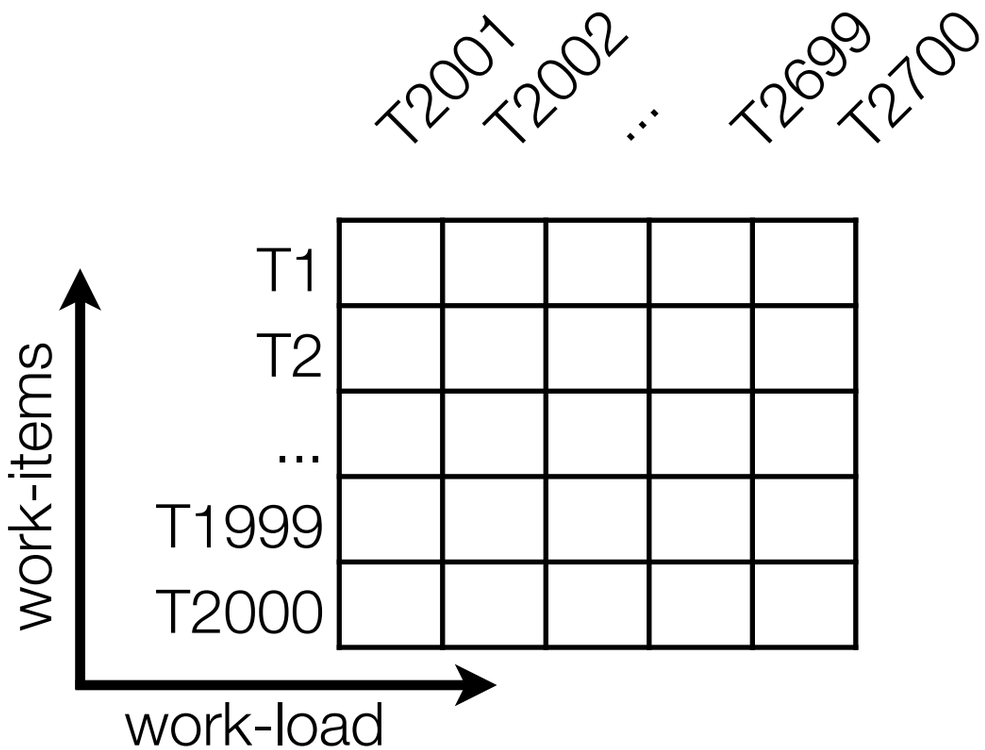
- alpha-matching
  - for each input triple one thread is created that checks, if that triple matches to one or more alpha-nodes (n triples → n threads)

Basic RETE algorithm (from [PBSZ13])

Parallelization: targeting massively parallel Hardware

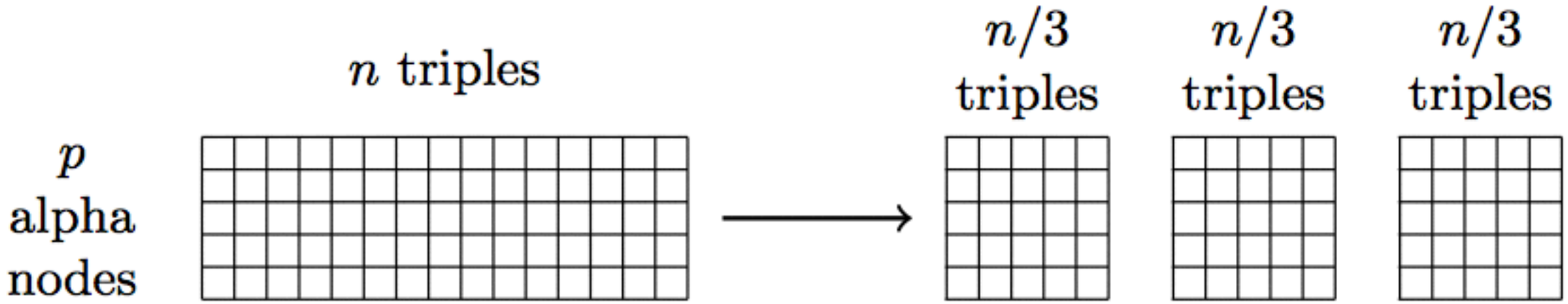
- alpha-matching
  - for each input triple one thread is created that checks, if that triple matches to one or more alpha-nodes (n triples → n threads)
- beta-matching
  - one thread for each match of one of the parent-nodes, that iterates through all matches of the second parent-node and checks for a match





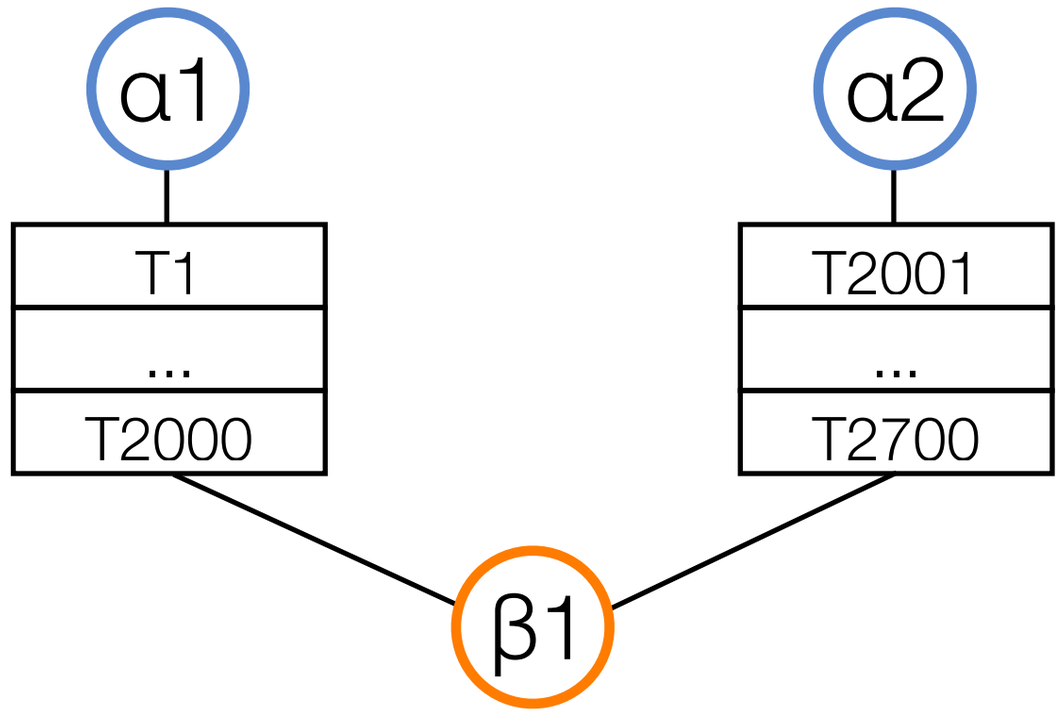
alpha-matching

- the workload can easily be partitioned into smaller chunks that can be processed independently
- the chunk size can be chosen with respect to the target device



beta-matching

- for beta-matching the workload cannot simply be divided
  - the triple-references in the working-memories need to be resolved
  - thus, all triples need to be available during that step
  - this would limit the size of processable data to the amount of triples, that fit into the memory of the GPU

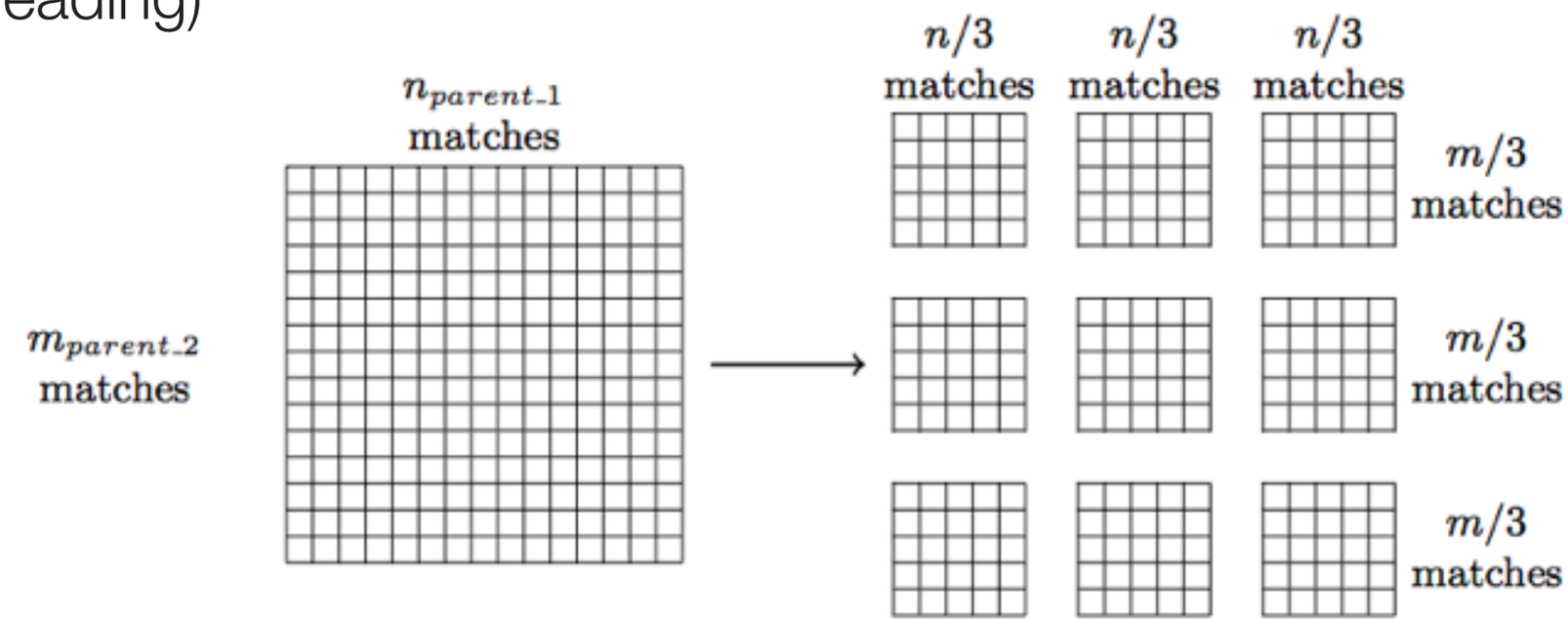


- to overcome this issue, we introduce a *triple-match as*:

  *A **triple-match m = (s,p,o,r)** is a quadruple with s=subject, p=predicate, o=object of a triple and r=triple reference (unique number, that is used for identification in the internal triple store).*

beta-matching II

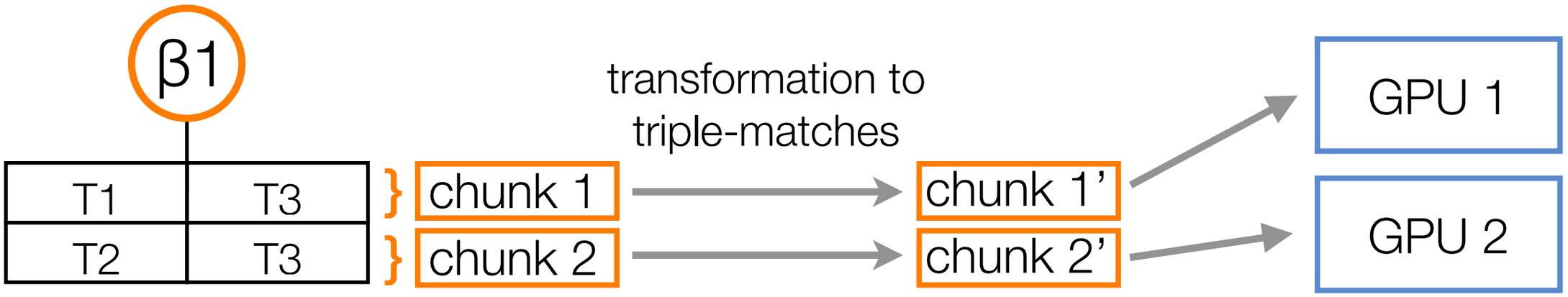
- working-memories need to be transformed to triple-matches which then are transferred to the GPU for beta-matching
  - the working-memories can be divided to smaller chunks
  - the transformation of the chunks can be done in parallel, too (using multithreading)



- because we are using triple-matches, no references need to be resolved on the GPU anymore and each chunk can be processed independently

Parallelization of rule-firing

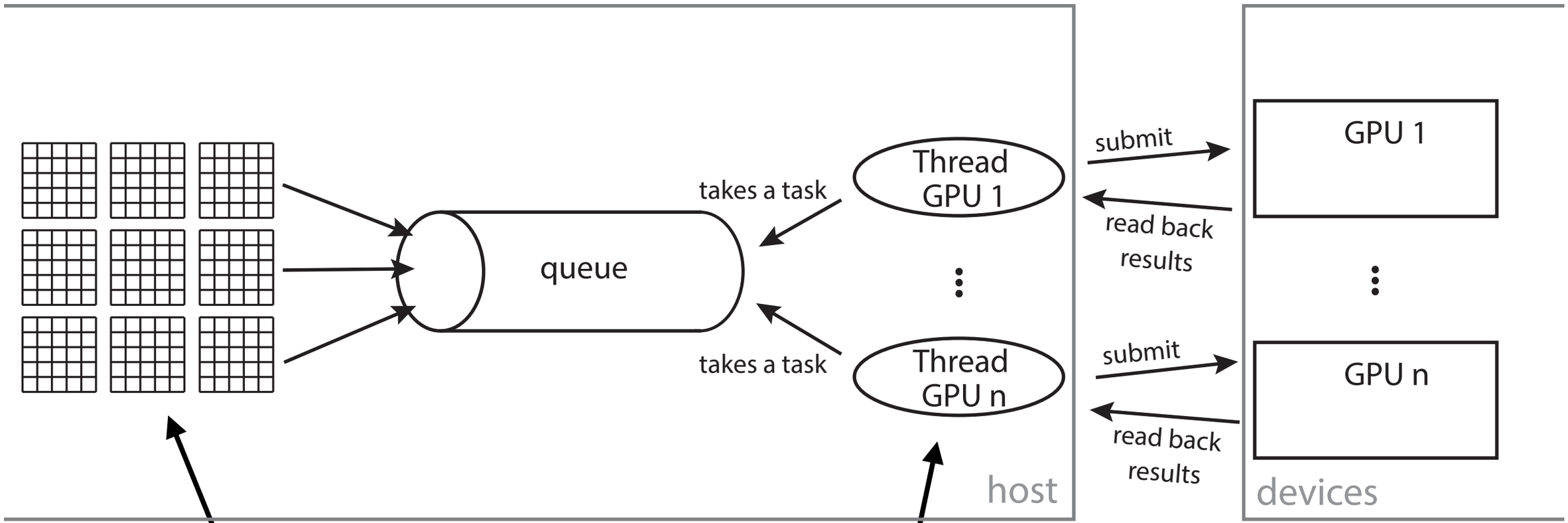
- like beta-matching, rule-firing can be performed on the GPU, too
  - working-memories need to be transformed to (chunked) triple-matches
  - triple-matches and a a set of instructions that define how new triples are created are transferred to the GPU



- the triple-store is responsible for rejecting invalid triples (duplicates and triples with a literal as subject)
- a simple reduction-concept based on triple-matches to reduce the amount of invalid triples was also introduced

Test environment: architecture

- the new concepts were implemented in the java-based reasoner from [PBSZ13]



chunks are prepared and submitted to the queue using java multithreading

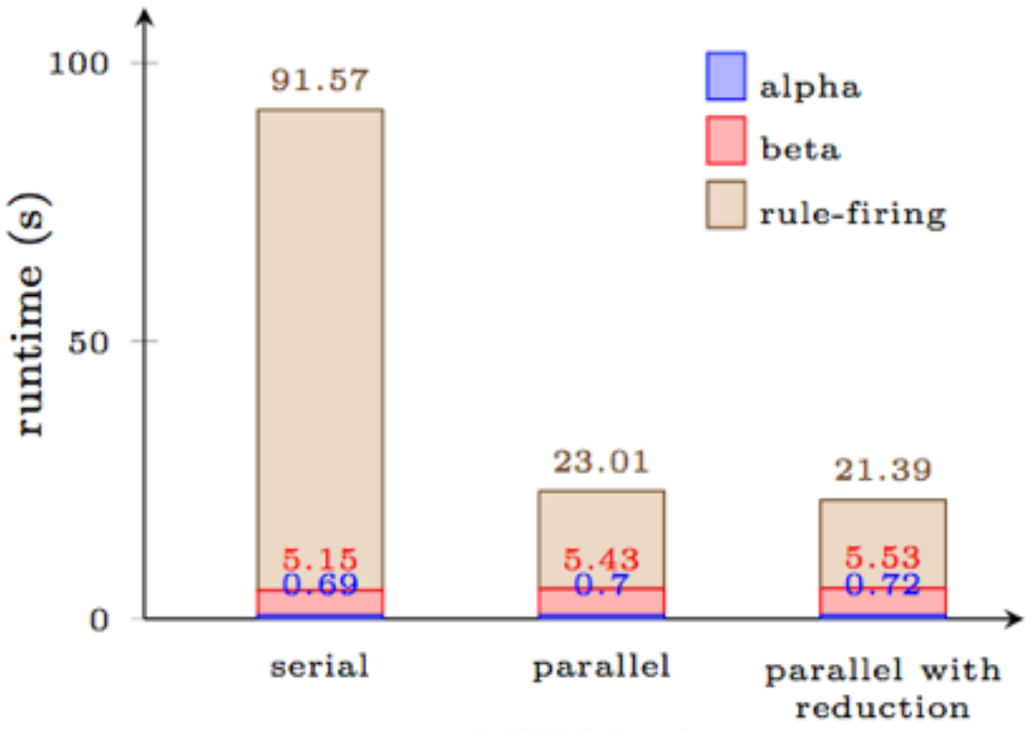
one thread for each GPU (each thread has exclusive rights to a GPU)

Test environment: tests

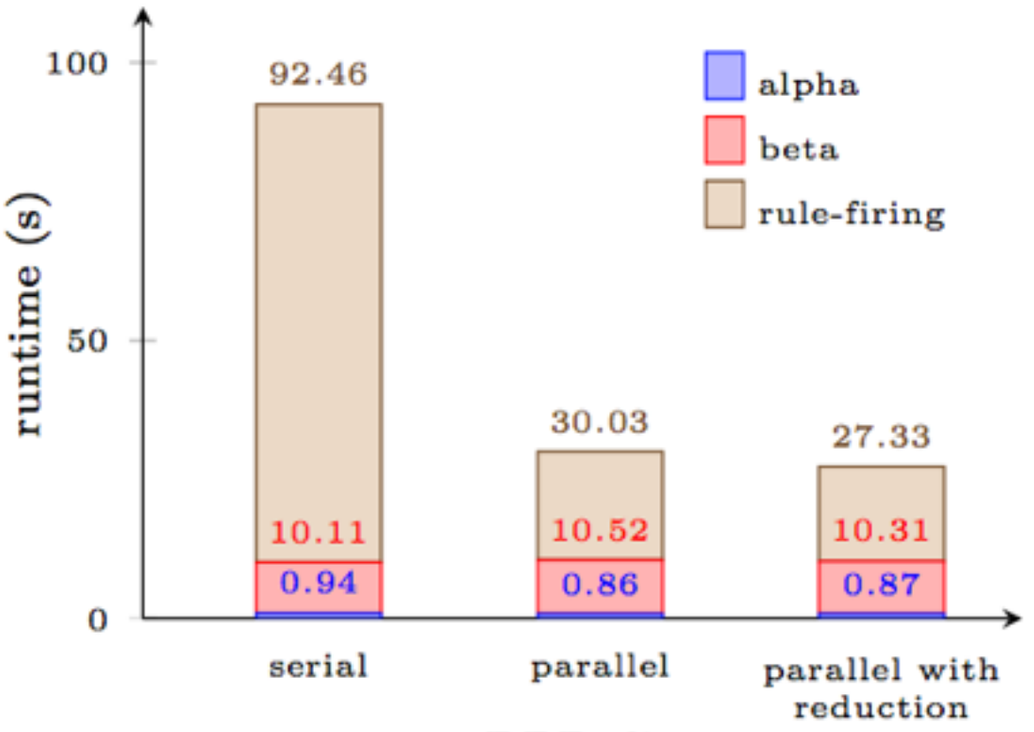
- Datasets:
  - Lehigh University Benchmark (LUBM): LUBM125 to LUBM8000
  - DBpedia scaled to full, 1/2nd, 1/4th, 1/8th, 1/16th, and 1/32nd
- Workstation with Ubuntu 12.04:
  - 2.0 GHz Intel Xeon processor with 6 cores
  - 64 GB memory
  - two AMD 7970 gaming graphic cards with 3GB of memory each

Parallel rule-firing and reduction of invalid triples

- Detailed reasoning time for applying the RDFS rule-set:



LUBM250 (~35M triples)

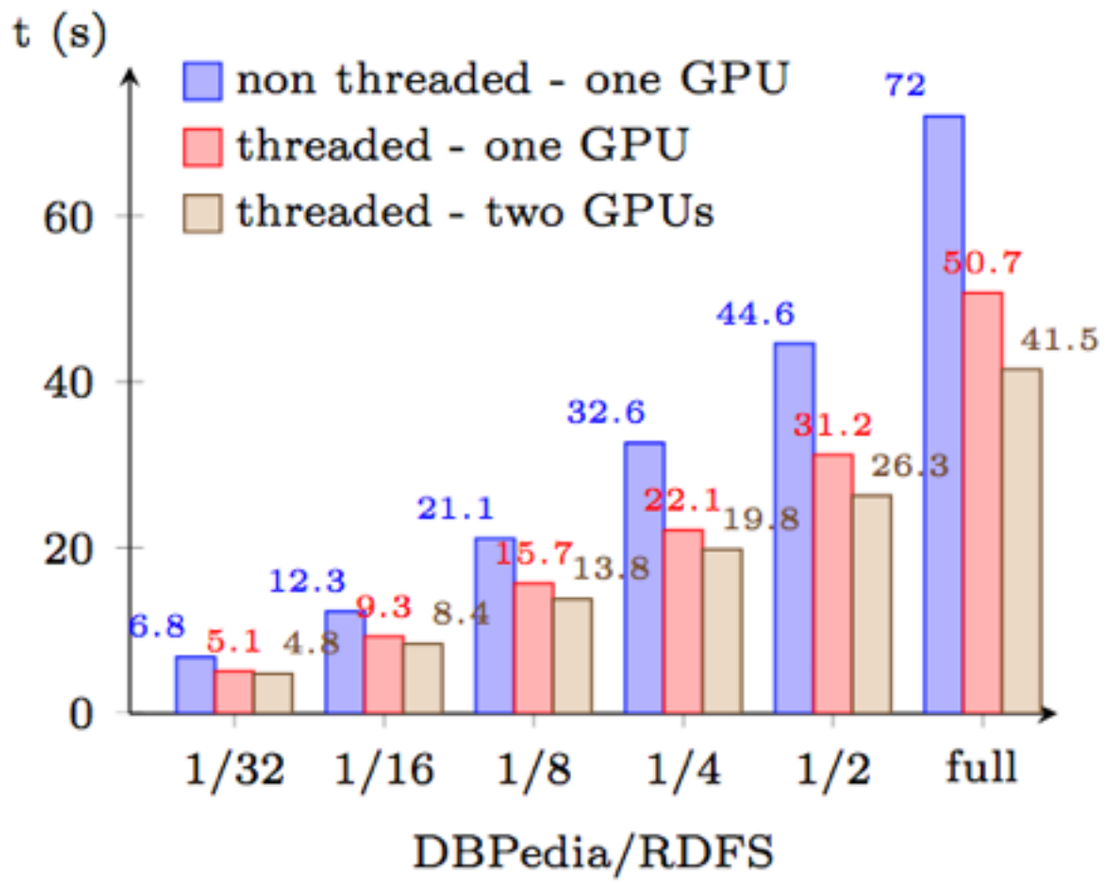


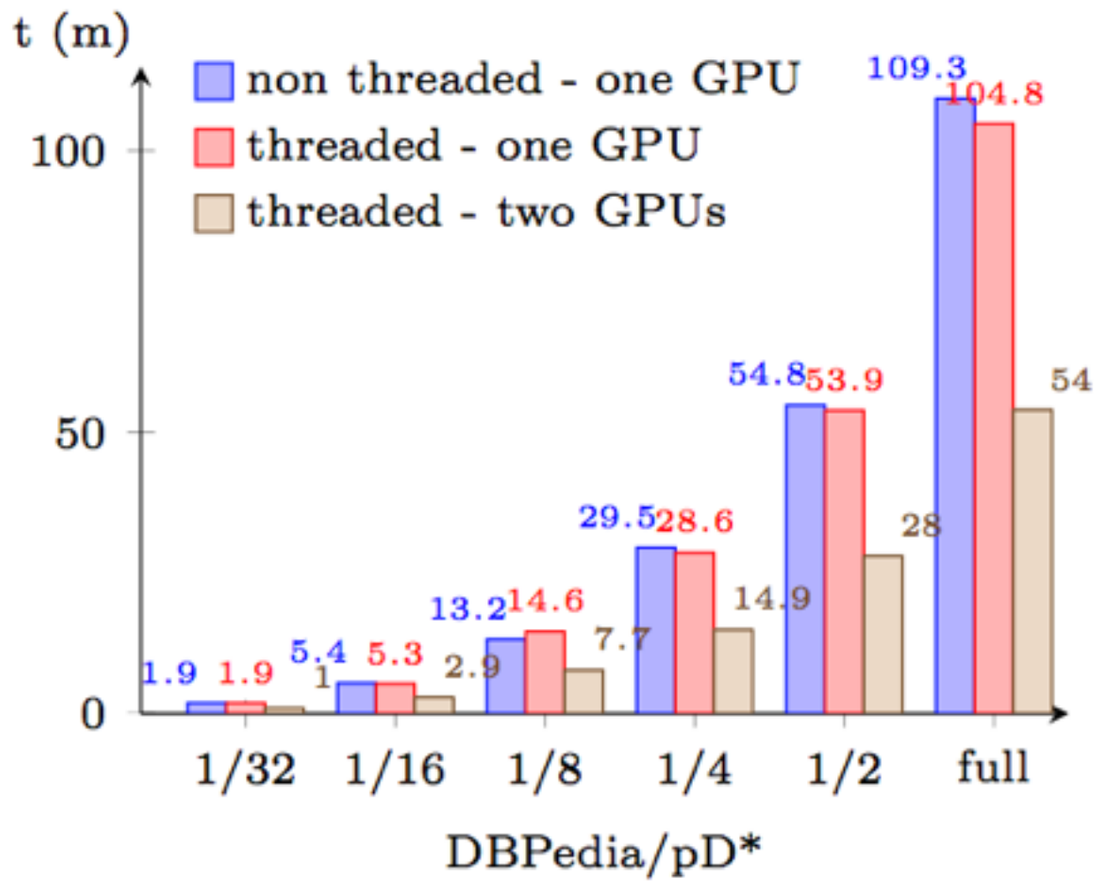
DBPedia (~41M triples)

- parallel rule-firing significantly improves the performance
- the proposed concept for reducing invalid triples improves the performance of another ~10%

Parallelization: using chunks and multiple GPUs

- Using RDFS (left) and pD* (right) on the DBPedia datasets

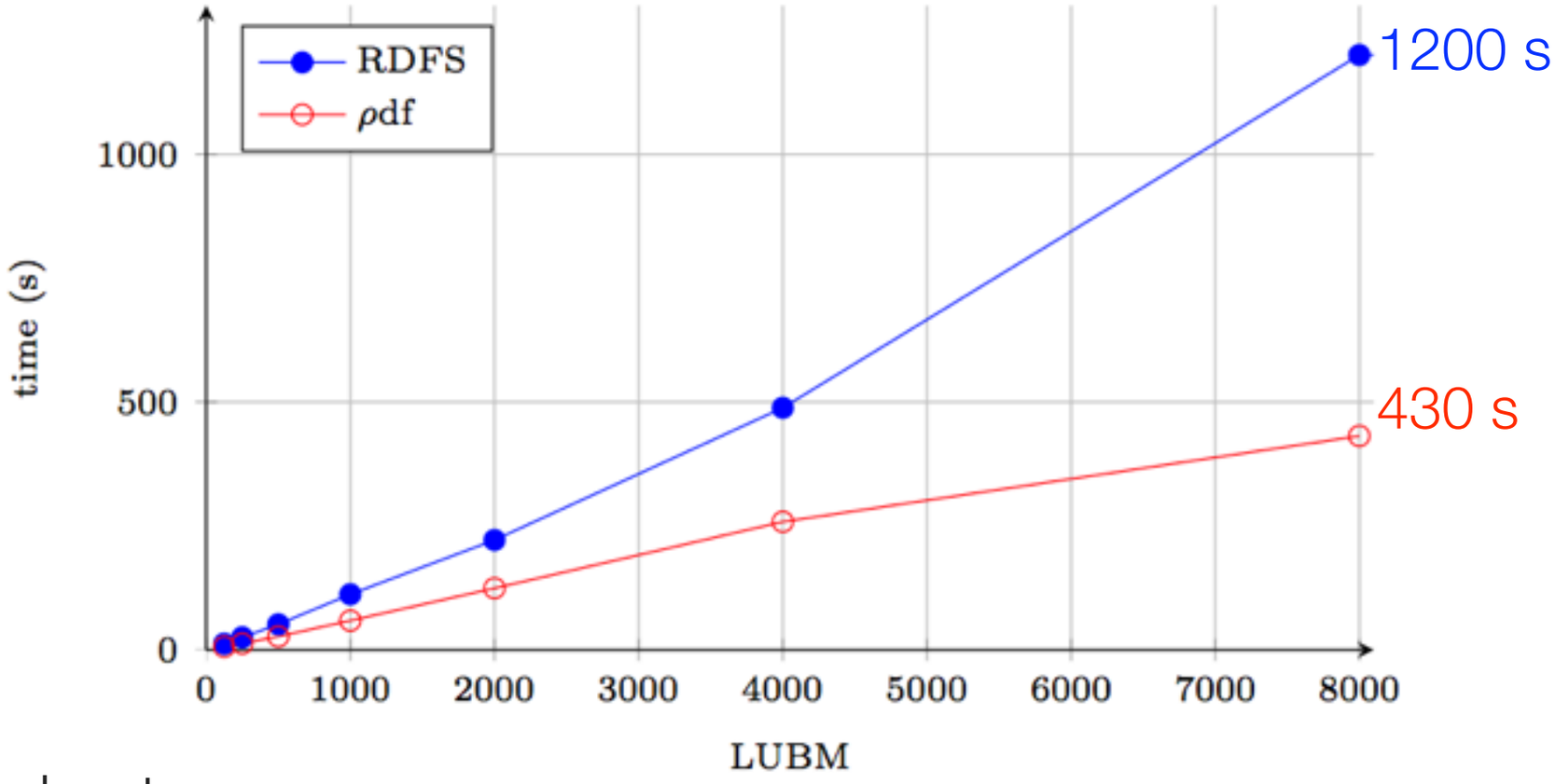




- RDFS reasoning benefits primarily from the thread-level parallelization
- pD* reasoning benefits significantly from the second GPU

Scaleability

- used Hardware: cloud-server with two Tesla M2090 GPUs and 192GB memory
- applying ρdf and RDFS to LUMB datasets from 17.6M to more than 1.1B triples



- max throughput:
  - ~ 2.7M triples/sec. for ρdf (WebPIE reported 2.1M triples/sec on 64 computing nodes)
  - ~ 1.4M triples/sec. for RDFS

Conclusion and future work

- we parallelized the RETE-algorithm in a way that
  - the preparation of the workload can be performed in parallel using multithreading
  - the workload can be distributed to multiple GPUs
- we were able to perform the reasoning on a dataset with more than 1B triples in about 430 seconds for ρdf and 1200 seconds for RDFS, on a single machine!
- the new limitation we reached is the main-memory of the computing node itself

- Future work will include:
  - integration of concepts to reduce the memory usage
  - distributing the workload not only to multiple GPUs, but also to multiple hosts equipped with GPUs

Fachhochschule Dortmund
University of Applied Sciences and Arts

Thank you for your attention!

contact:

Martin Peters
martin.peters@fh-dortmund.de