On Matching Large Life Science Ontologies in Parallel

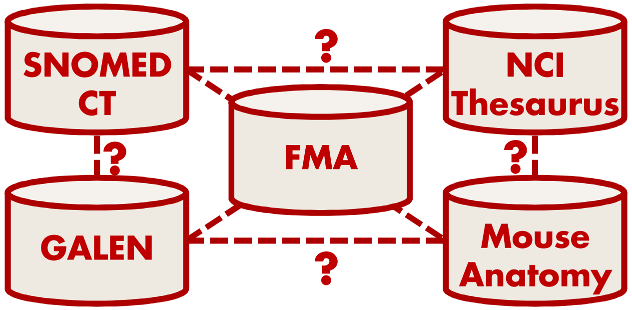
Anika Groß, Michael Hartung, Toralf Kirsten, Erhard Rahm

Database Group Leipzig
http://dbs.uni-leipzig.de

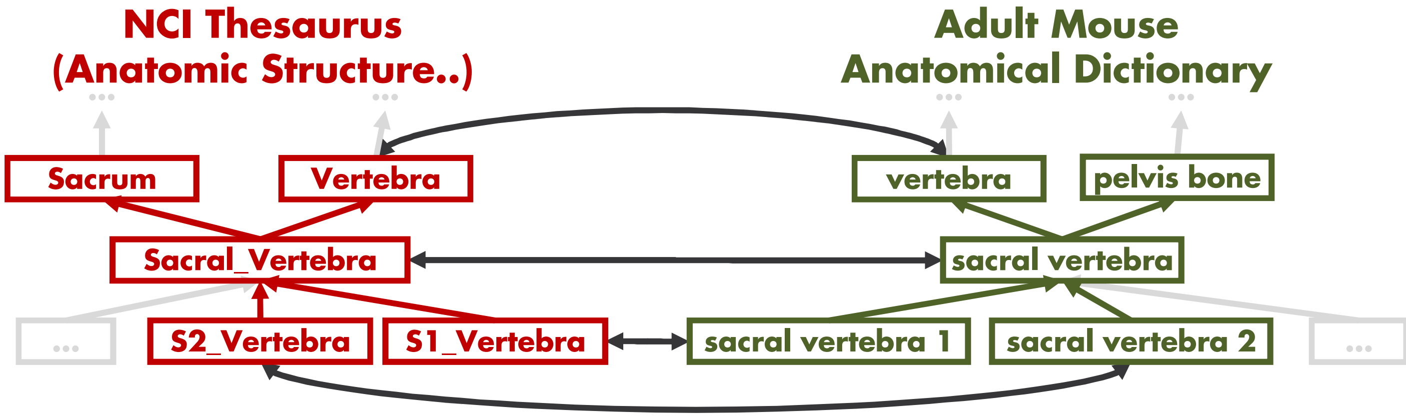
Interdisciplinary Centre for Bioinformatics
http://www.izbi.uni-leipzig.de

Ontology Mappings

- Ontologies are often highly related
- Identify overlapping information
- Crucial for data integration, enhanced data analysis across ontologies, ontology merging, …



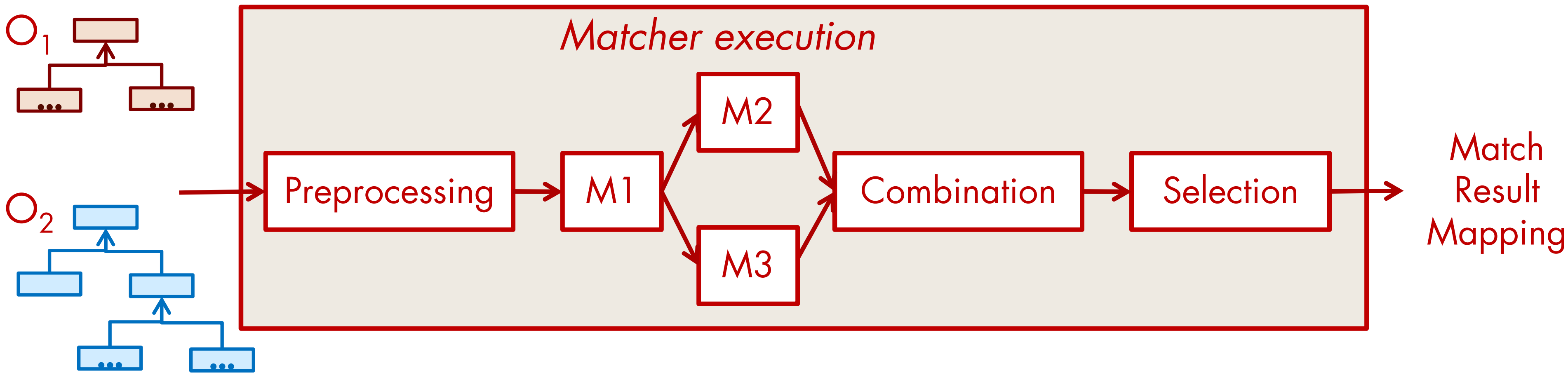
- Ontology Mapping = set of semantic correspondences between concepts of different ontologies



- Manual creation is complex or even infeasible
- (Semi-) automatic determination of similarities between ontology elements to find correspondences

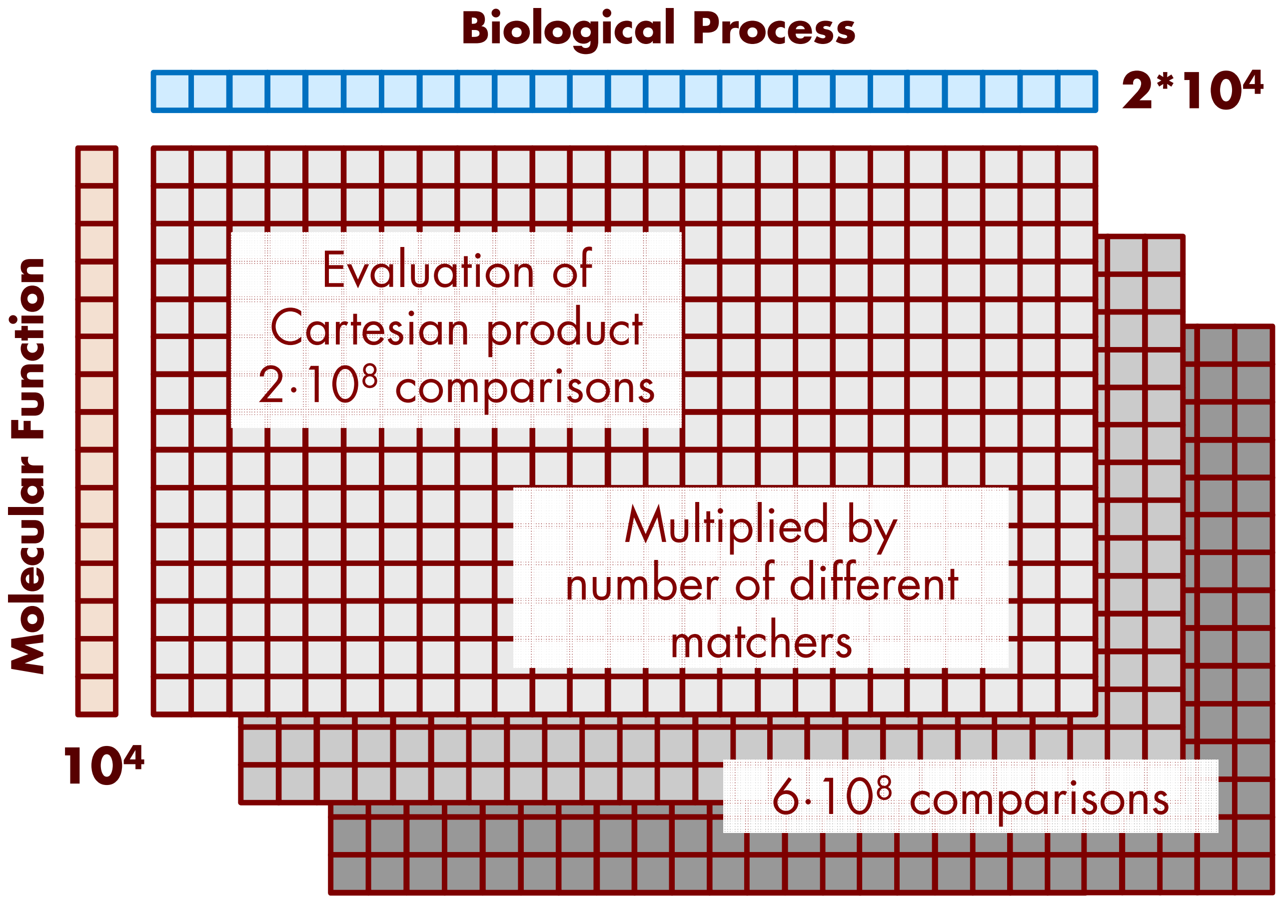
Ontology Matching

- Use metadata (element-, structure-level), instance data
- High quality mappings
- Combined execution of several matchers in more complex **match workflows**



- **Time consuming and memory intensive task**

Example – Match Sub-Ontologies of Gene Ontology



- → Long execution times, high memory requirements
- → Reasonable to improve efficiency

How to deal with Performance and Memory issues?

→ **Parallelization**

- Parallel execution of ontology matching on multiple compute nodes
- Use the broad availability of multi-core systems
- Reduce execution time, memory requirements

→ **Further methods**

- Reduction of search space: Avoid evaluation of Cartesian product using e.g. early pruning, cluster- or fragment-based methods
- Reuse of match results: Save recomputation (ontology evolution)
- ...

→ **Combination of approaches**

CONTRIBUTIONS

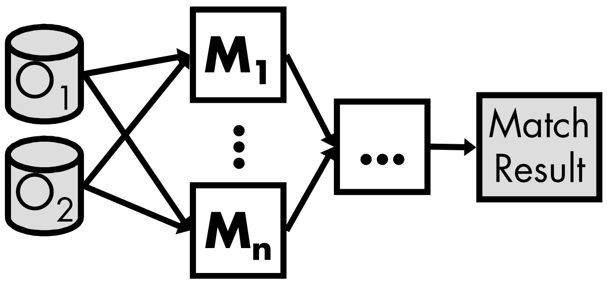
- Two strategies for parallel ontology matching
  - Inter-matcher parallelization: execute independent matchers in parallel
  - Intra-matcher parallelization: internal parallelization of matchers based on partitioning of the ontologies
- Parallelization of different kinds of matchers
  - Element-level, structure-level, instance-based matchers
- Implementation and evaluation
  - Distributed infrastructure for parallel ontology matching
  - Evaluation of the approaches for matching large life science ontologies

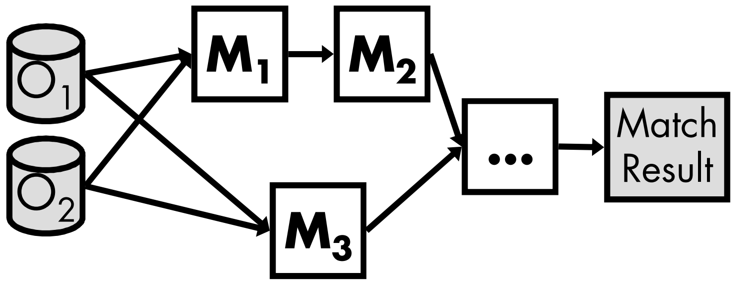
OVERVIEW

- Motivation
- Parallelization Strategies
  - Inter-matcher Parallelization
  - Intra-matcher Parallelization
- Infrastructure
- Evaluation
- Conclusion & Future Work

INTER-MATCHER PARALLELIZATION

- Parallel execution of independently executable matchers
- Process matchers on different cores or computing nodes





\+ Improve execution time up to factor *n* (*n*=|matchers|)

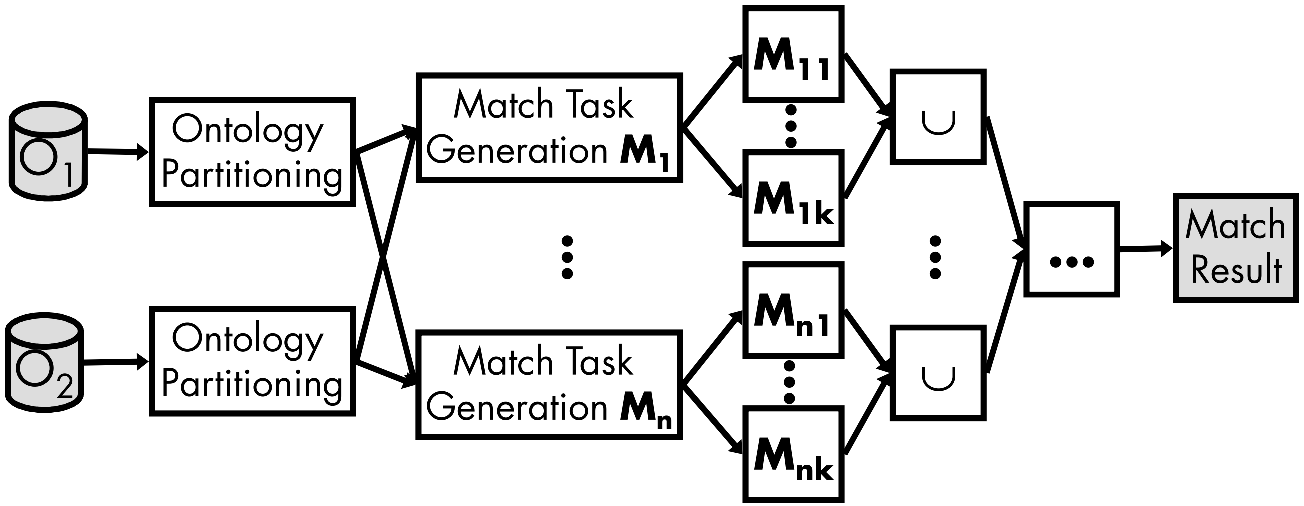
– Limited degree of parallelization (|independently executable matchers|)

– Slowest matcher limits the achievable speedup

– Memory requirements are not reduced since matchers evaluate complete ontologies

Intra-Matcher Parallelization

- Internal decomposition of individual matchers or matcher parts
- Partitioning the input data (ontologies)
- Many small match tasks can be executed in parallel

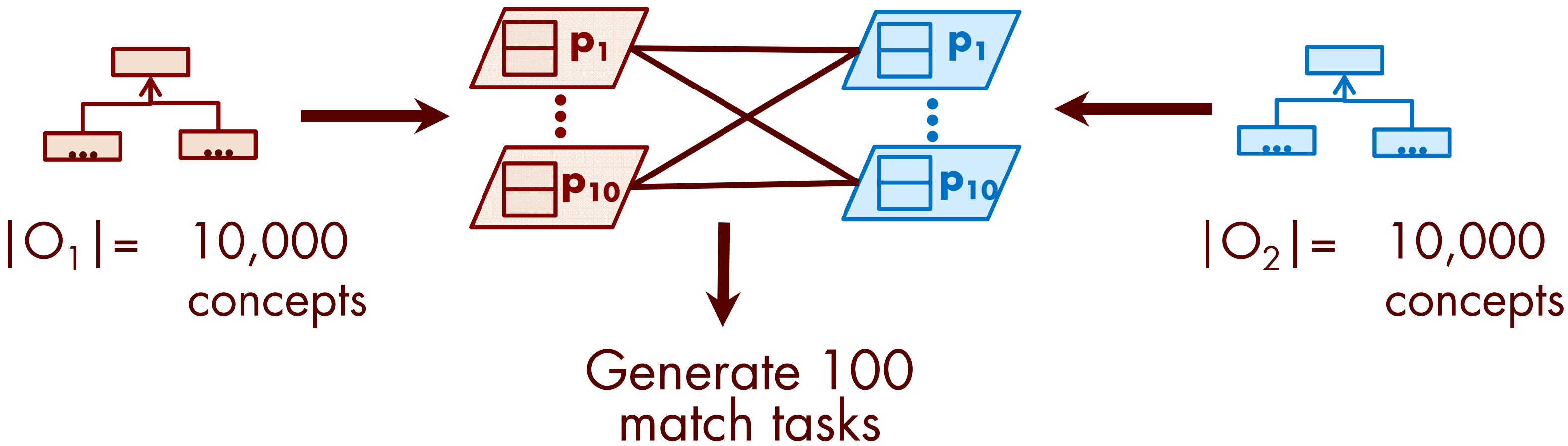


+ Match tasks of limited complexity
  → reduced memory and processing requirements
+ Applicable to sequential and independently executable matchers

Size-Based Partitioning

- Enable parallel matching of Cartesian product
- Partition input ontologies into partitions of equal size (|concepts|)

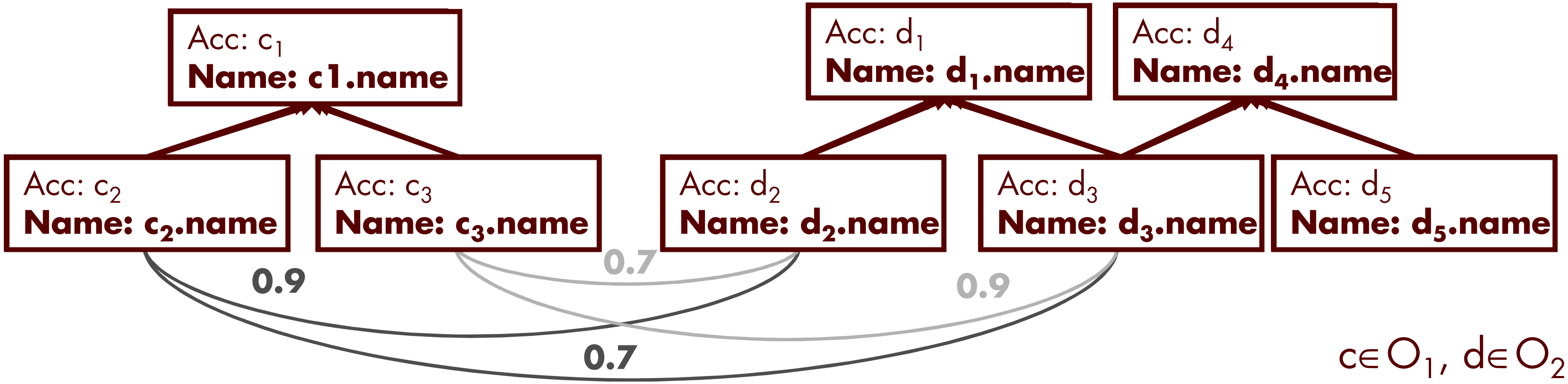
- Each match task matches one O_1 against one O_2 partition



+ Scalable to large ontologies
+ Good load balancing
+ Optimizes performance without sacrificing match quality
+ Applicable to element-level, structure-level and instance-based matchers

Parallelization of Element-level Matchers

- Needed information is directly associated with concepts themselves
- Attribute values like names, synonyms …



$c \in O_1$, $d \in O_2$

→ Match-information is retained

→ Easy partitioning of ontologies into subsets of concepts for element-level matchers

Parallelization of Structure-level Matchers

- Needed information is not provided by concepts themselves
- Use information from the neighborhood of concepts, e.g. children, siblings, namePath, …

Approach

- Extend the concept-level information within special *multi-valued context attributes*, e.g. *Child*, *NamePath*, …
- Preprocessing step of linear effort

→ Size-based partitioning as for element-level matching

- Note: parallelization of similarity propagation algorithms is more difficult (need the whole structural information)

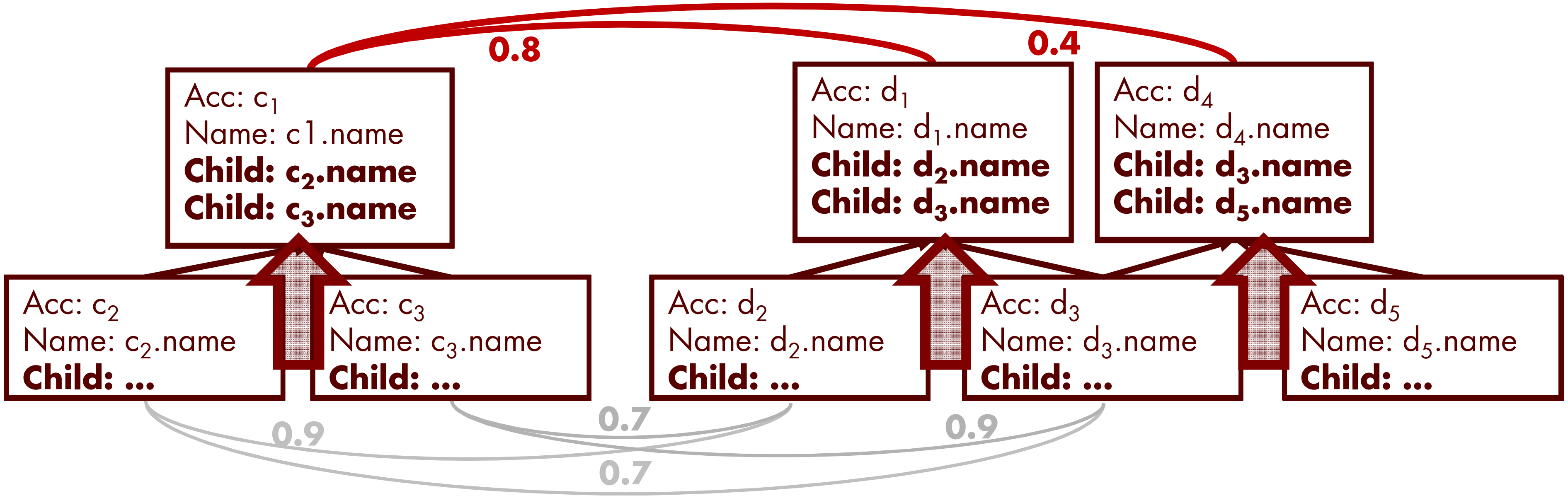
→ Only parallelization of the initial matcher

EXAMPLE: CHILD MATCHER

- Use a multi-valued *Child* context attribute: get name values of child concepts in a preprocessing step
- Average element similarity between children of concepts

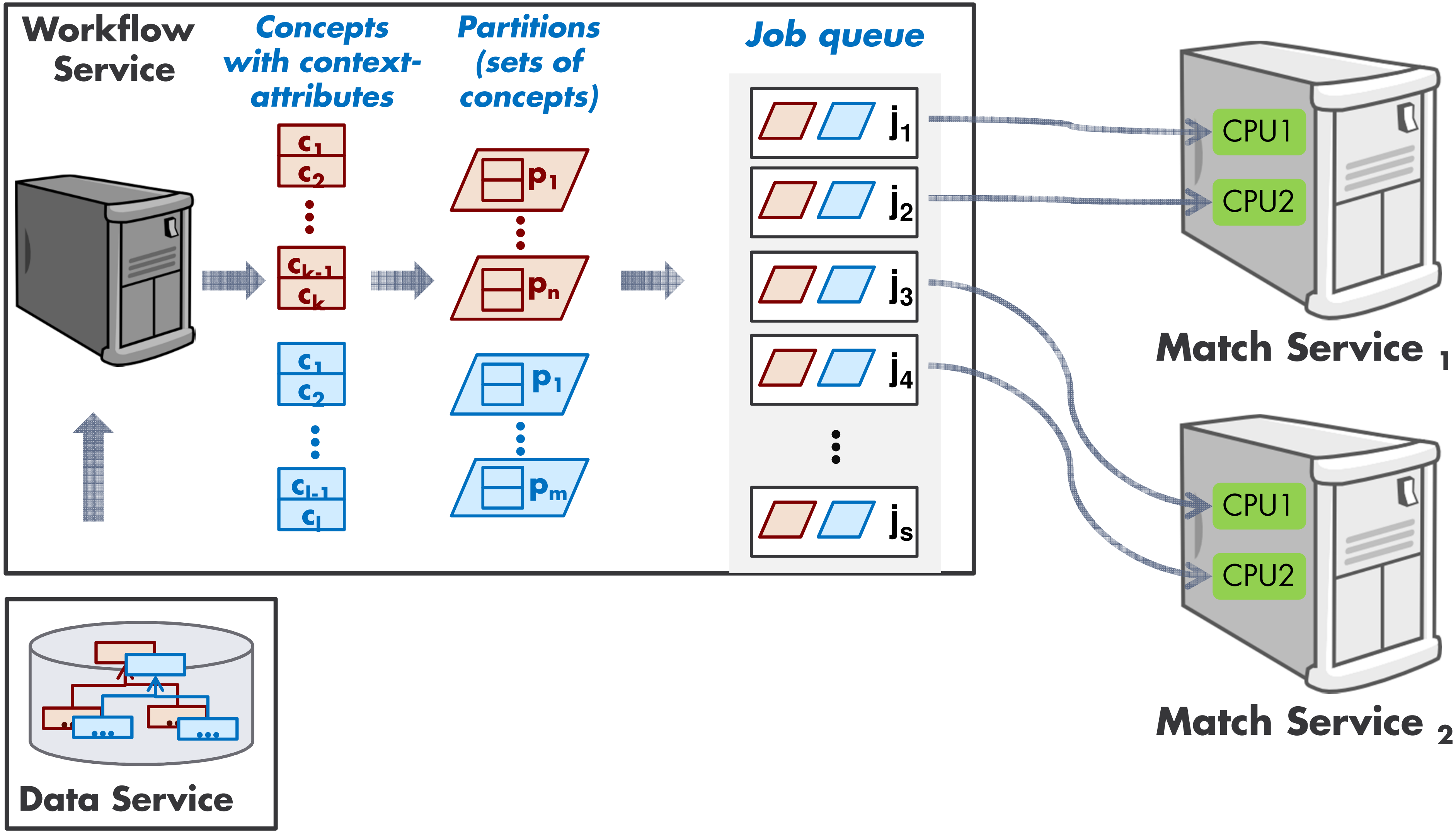
→Compare all their *Child* attributes

corr(c,d)	***sim$_{Children}$(c,d)***
$c_1 - d_1$	(0.9 + 0.7 + 0.7 + 0.9) / (2·2) = **0.8**
$c_1 - d_4$	(0.7 + 0 + 0 + 0.9) / (2·2) = **0.4**

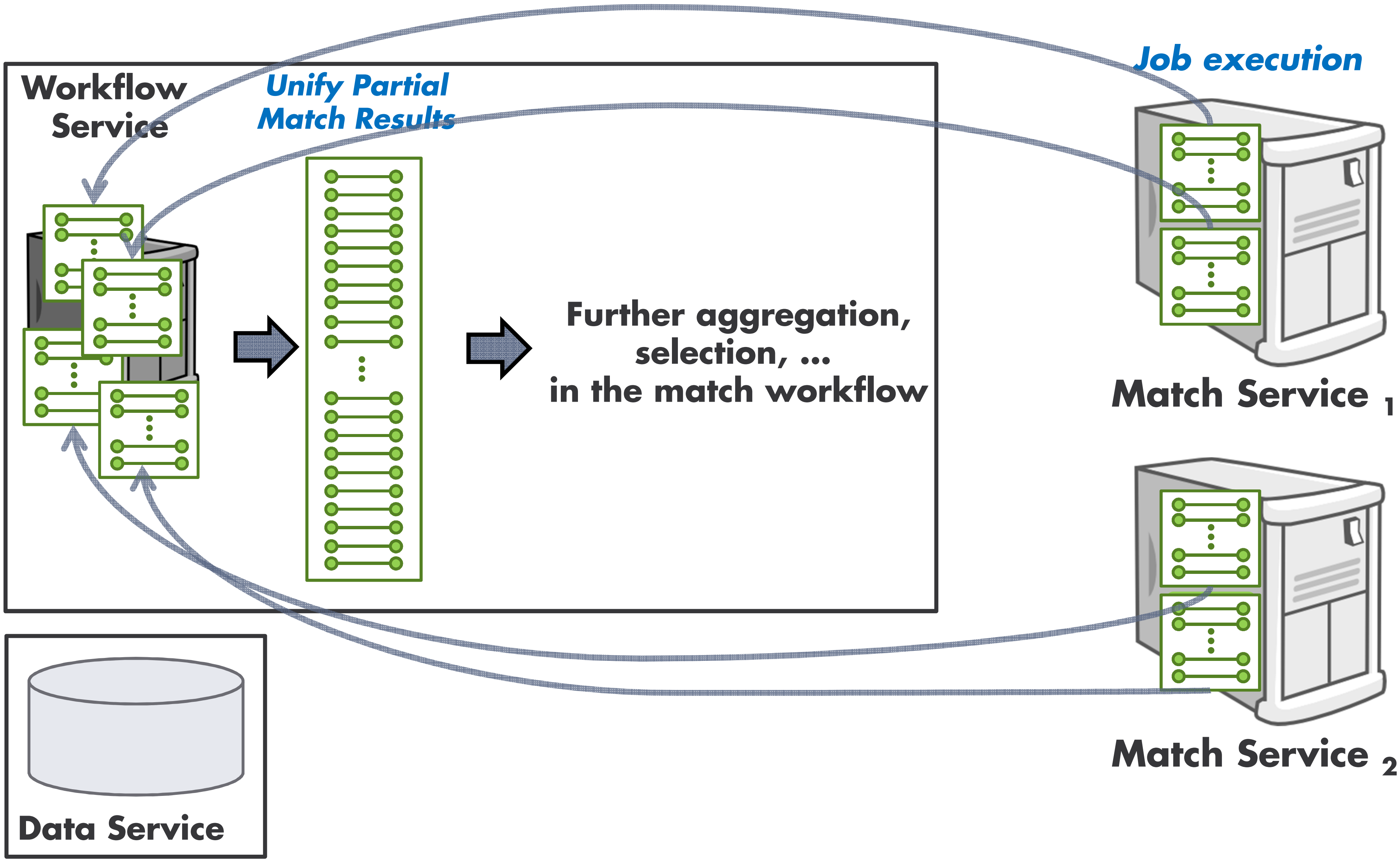


- Similarly applicable for other local context matchers, e.g. namePath

INFRASTRUCTURE INTRA-MATCHER PARALLELIZATION EXAMPLE



INFRASTRUCTURE – INTRA-MATCHER PARALLELIZATION EXAMPLE



Evaluation

- Two match problems to evaluate efficiency (execution times)

Adult Mouse Anatomical Dictionary · **NCI Thesaurus Anatomic Structure or Substance** — **Medium Scale**

~2,700 · ~3,300 = 9 · 10⁶ comparisons

Molecular Functions · **Biological Processes** — **Large Scale**

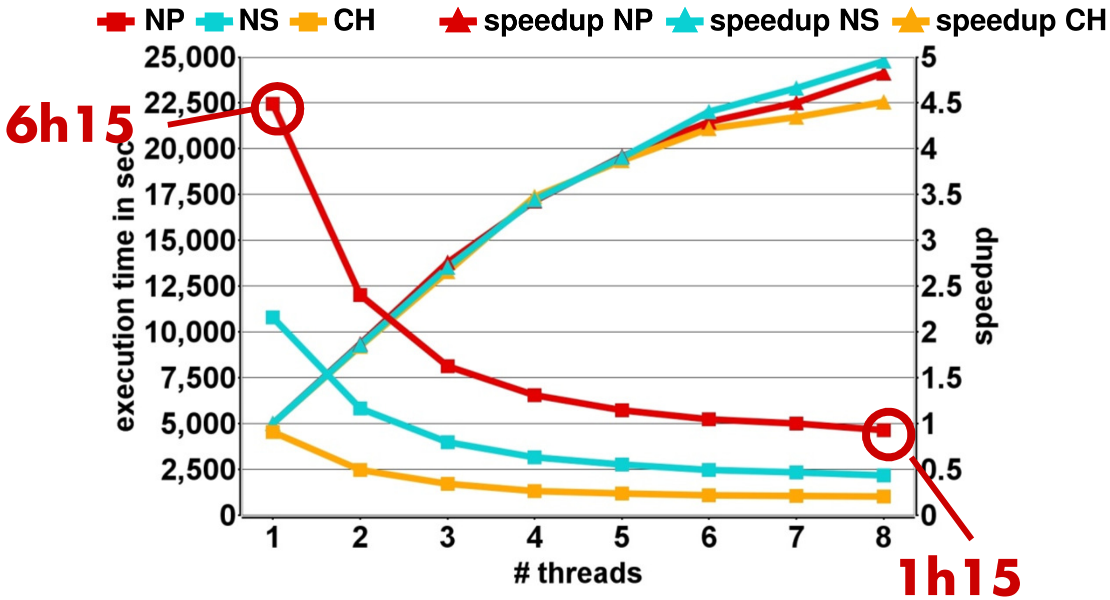
~9,400 · ~17,100 = 1.6 · 10⁸ comparisons

- Matchers (element and structure-level)
  - NameSynonym (NS) *Max TriGram similarity for the name and multi-valued synonym attr.*
  - Children (CH) *Avg TriGram similarity on the context attributes Child*
  - NamePath (NP) *Avg TriGram similarity on the context attributes NamePath*

- Computing environment
  Four nodes consisting of four cores (up to 16 cores)

INTRA-MATCHER PARALLELIZATION OF INDIVIDUAL MATCHERS

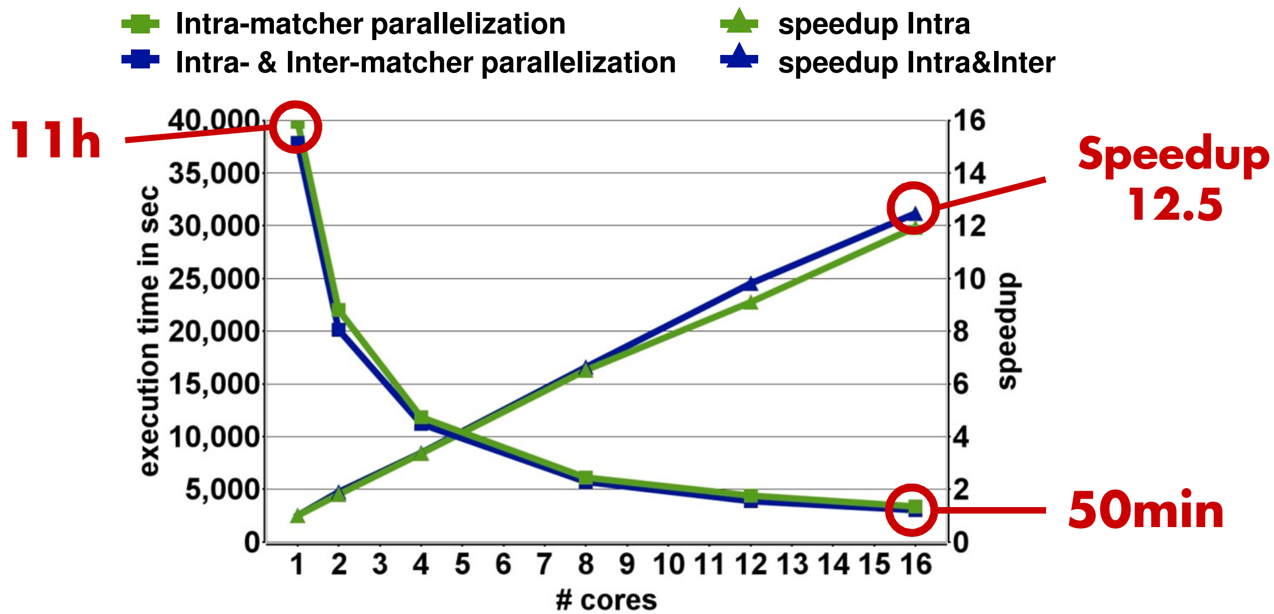
- 1 node, 4 cores, 8 threads



- ↑ degree of parallelization → ↓ execution times
  - NP - most expensive, long concatenated names, multiple inheritance
  - NS - many synonyms per concept in GO
- Almost linear speedup for up to 4 threads

Parallelization Strategies

- 4 nodes, 16 cores; matcher combination of NP, CH, NS



- Parallelization strategies benefit from multiple threads/cores
- Combined approach slightly better, because execution delays between matchers are avoided

→ Intra and the combined approach are very effective and thus especially valuable for parallel ontology matching

Conclusions and Future Work

- General, flexible strategies for parallel ontology matching on multiple compute nodes to improve efficiency (**inter- and intra-matcher parallelization**)
- **Size-based partitioning** enabled good load balancing, scalable, reduced memory consumptions, quality not affected
- Parallelization of element-level, structure-level, instance-based matchers using **multi-valued context attributes**
- Implemented a distributed infrastructure, evaluation for large life science ontology match problems

- Investigate parallel ontology matching for additional matchers, evaluate effectiveness
- Combine parallelization with advanced fragmentation strategies

Thank You For Your Attention