GRAPH PROCESSING AND PATTERN MATCHING

SIGMo
High-Throughput Batched Subgraph Isomorphism on GPUs for Molecular Matching

Antonio De Caro[1], Gennaro Cordasco[1], Federico Ficarelli[2], Biagio Cosenza[1]

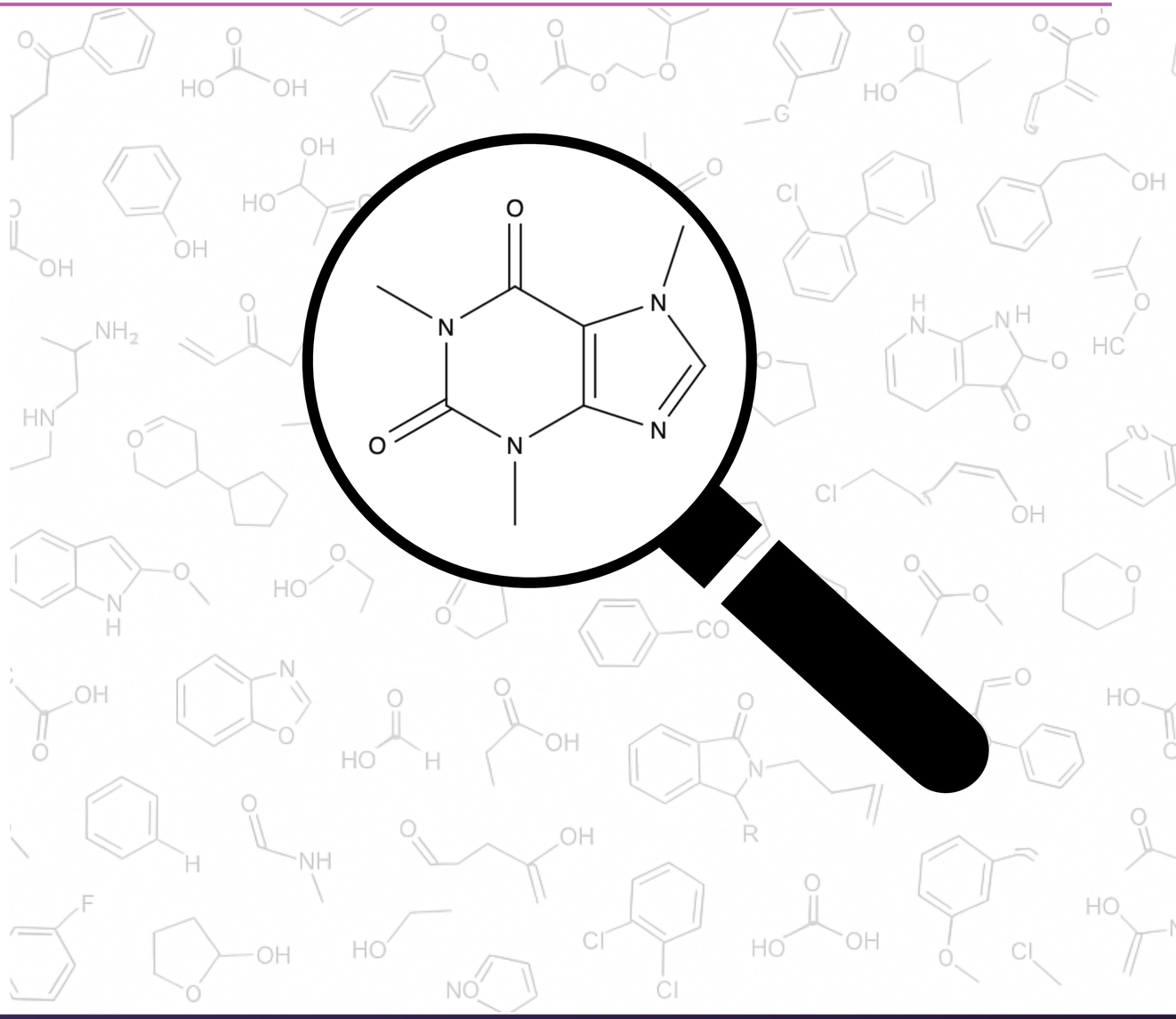
1. University of Salerno • Fisciano (SA), Italy

2. CINECA • Casalecchio di Reno (BO), Italy

SC25 ST. LOUIS hpc ignites.

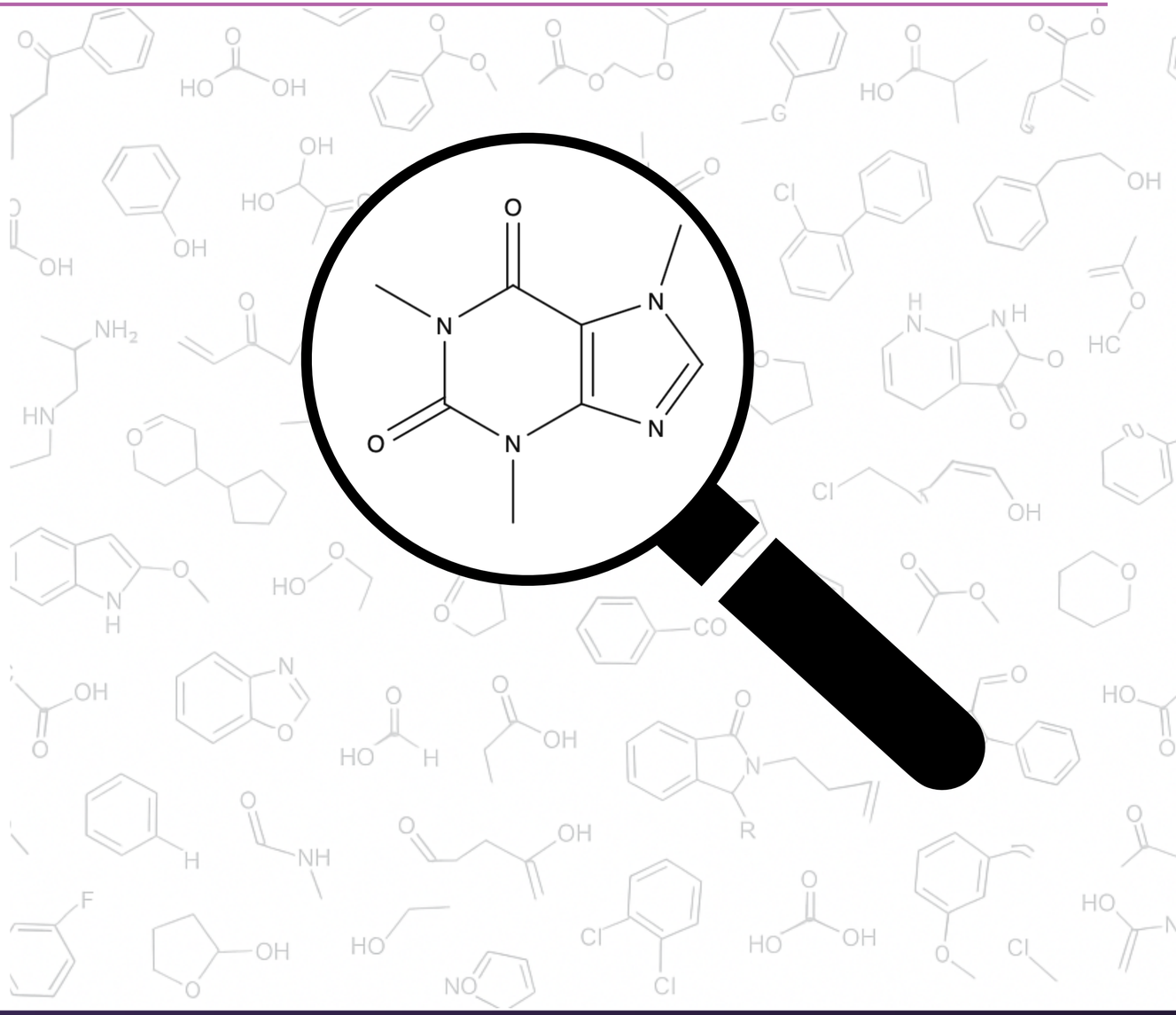
Why Drug Discovery Needs HPC

- New drugs are essential to fight emerging diseases
- Traditional methods are resource and labor intensive
- Computational methods can be exploited to lighten this burden, but:



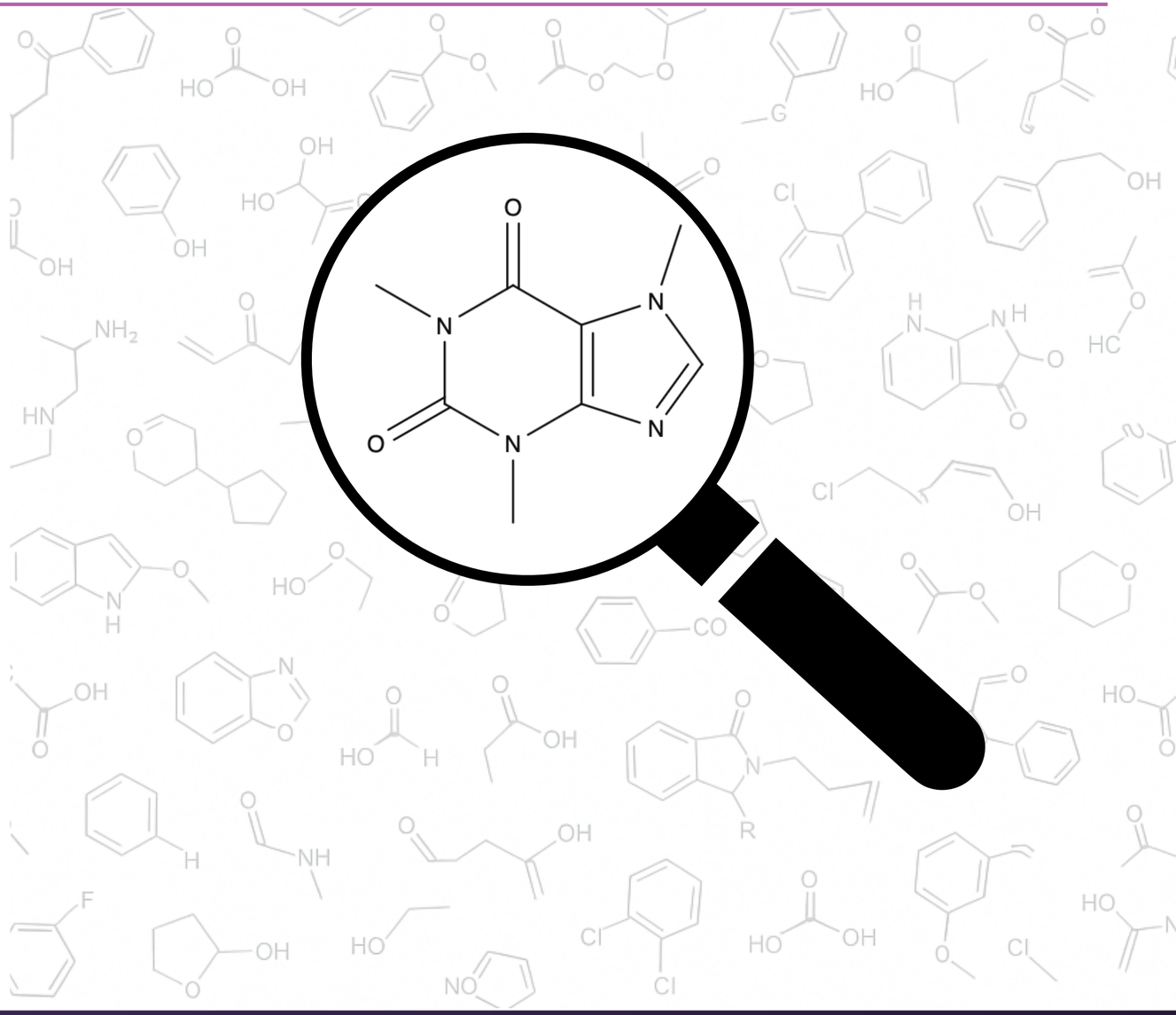
Why Drug Discovery Needs HPC

- New drugs are essential to fight emerging diseases
- Traditional methods are resource and labor intensive
- Computational methods can be exploited to lighten this burden, but:
  - The search space? **Millions to Billions** of molecules



Why Drug Discovery Needs HPC

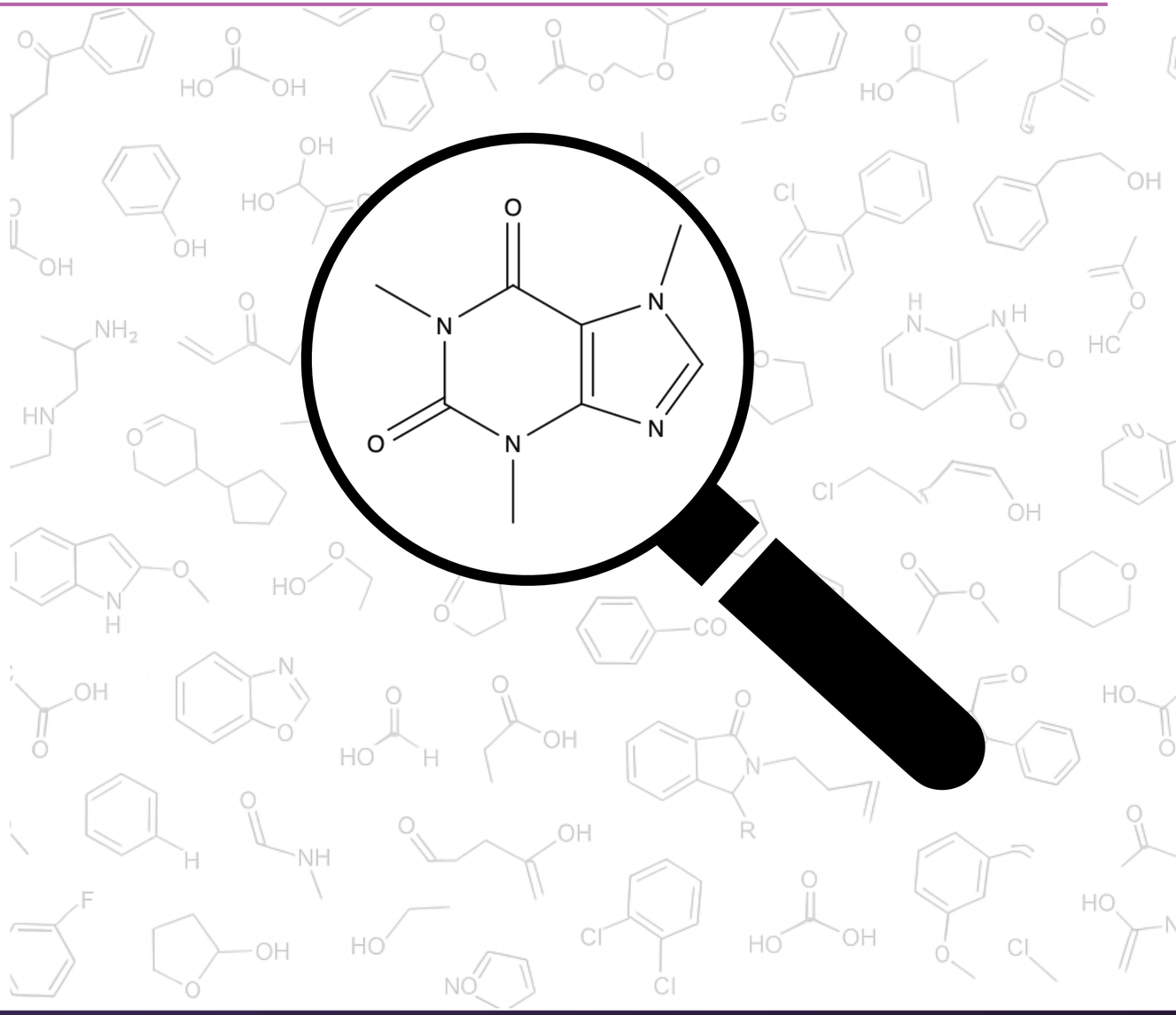
- New drugs are essential to fight emerging diseases
- Traditional methods are resource and labor intensive
- Computational methods can be exploited to lighten this burden, but:
  - The search space? **Millions to Billions** of molecules
  - Hundreds of **functional patterns**



Why Drug Discovery Needs HPC

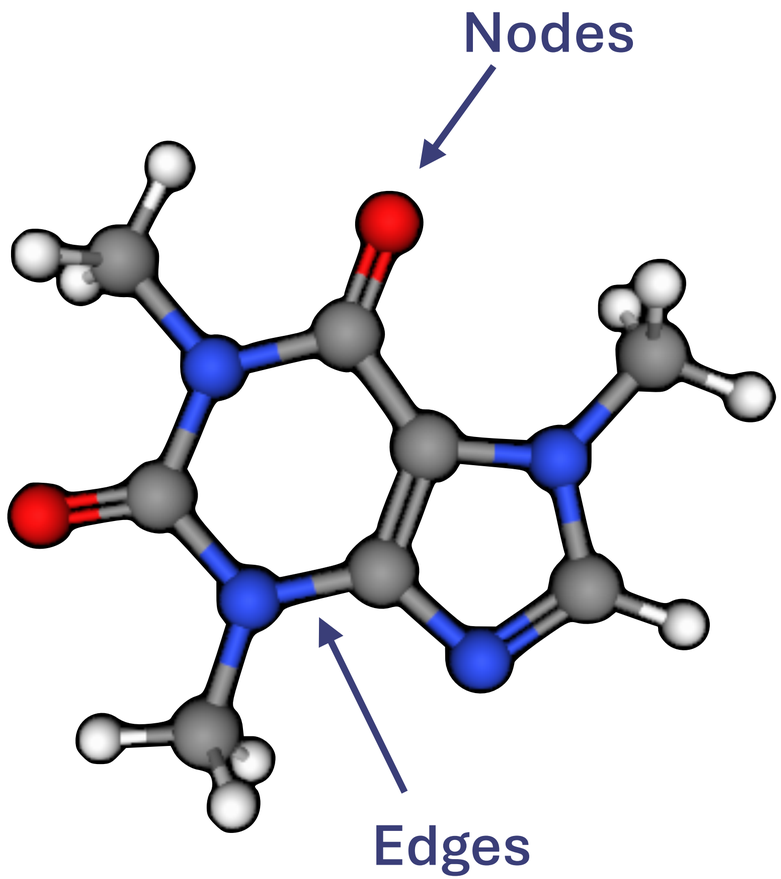
- New drugs are essential to fight emerging diseases
- Traditional methods are resource and labor intensive
- Computational methods can be exploited to lighten this burden, but:
  - The search space? **Millions to Billions** of molecules
  - Hundreds of **functional patterns**

➡ **Trillions of comparisons!**



Molecular Matching = Subgraph Isomorphism (But Constrained)

- Graphs can naturally be used to model molecules
- Finding whether a pattern occurs in a molecule reduces to solving the **Subgraph Isomorphism** problem (**NP-Complete**)

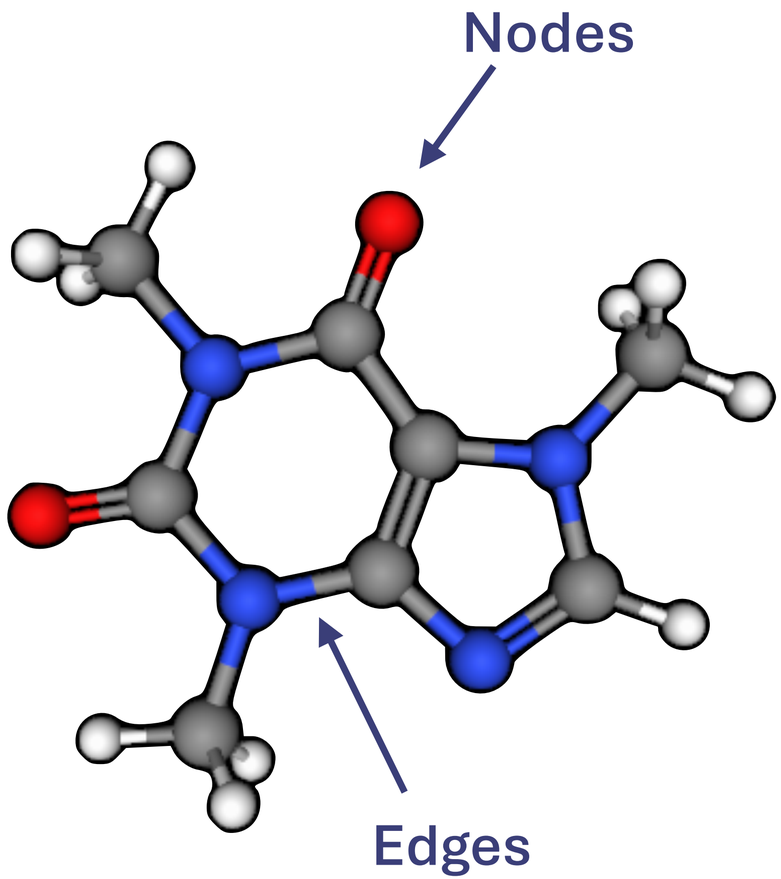


Molecular Matching = Subgraph Isomorphism (But Constrained)

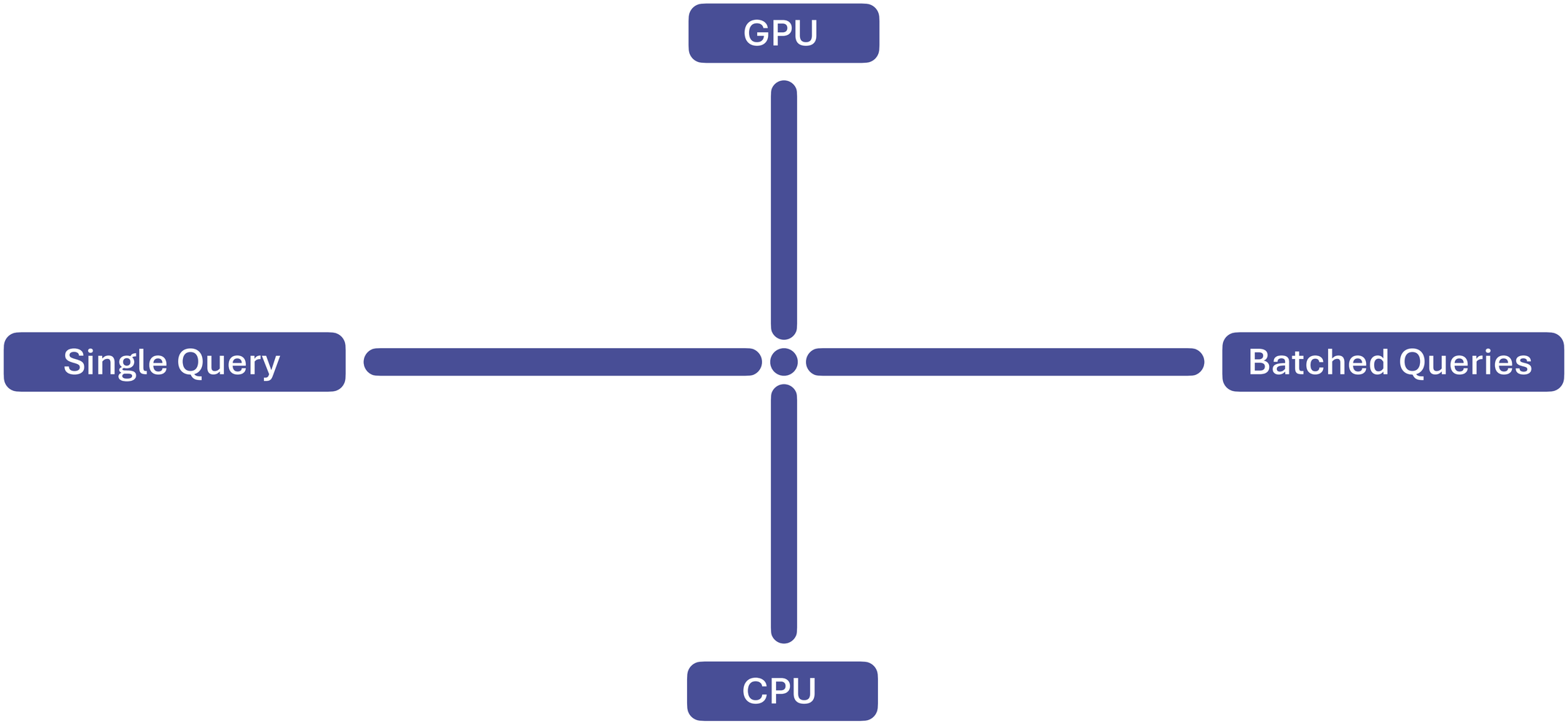
- Graphs can naturally be used to model molecules
- Finding whether a pattern occurs in a molecule reduces to solving the **Subgraph Isomorphism** problem (**NP-Complete**)

Molecular graphs are not general-purpose graphs due to physics constraints

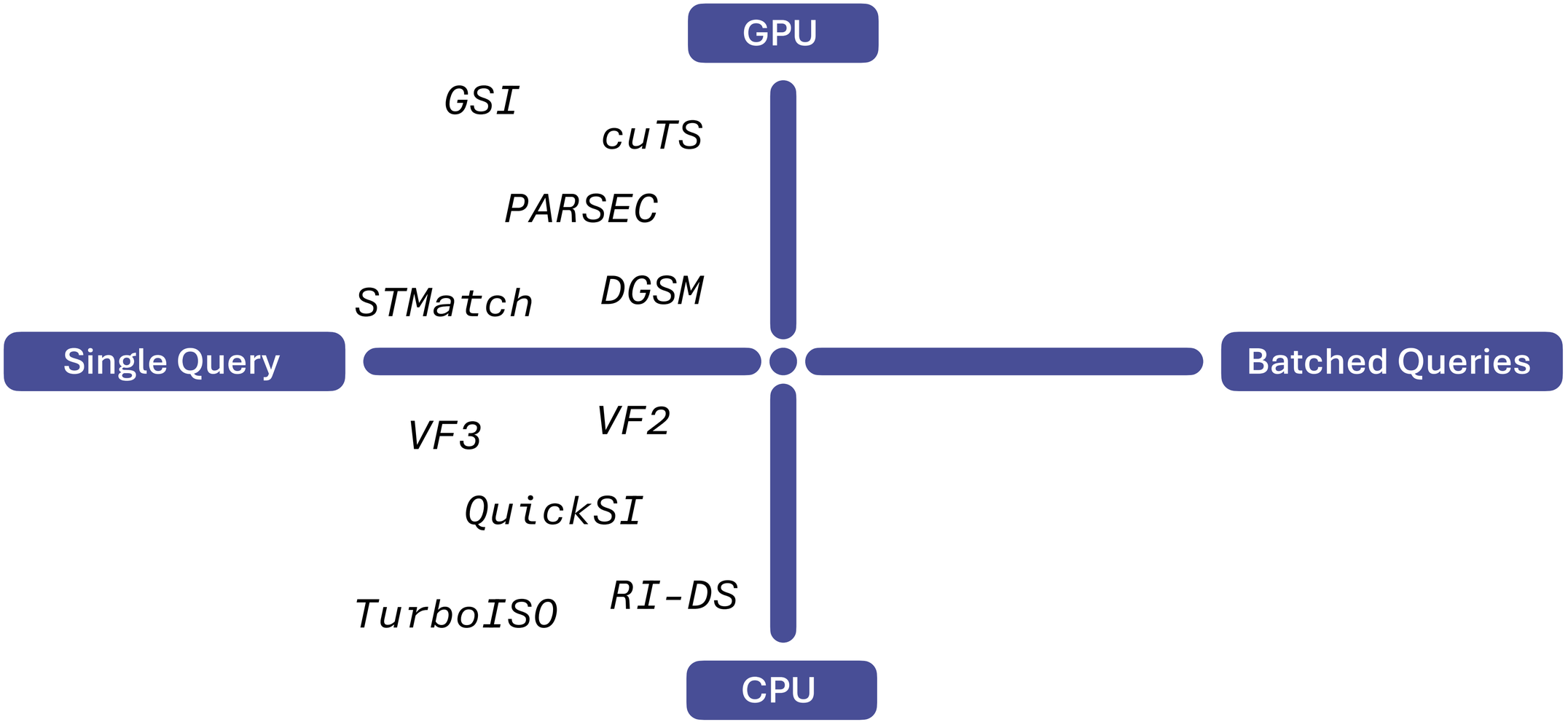
Low degree (≤ 4), few labels (atomic table), small diameter, sparse, ...



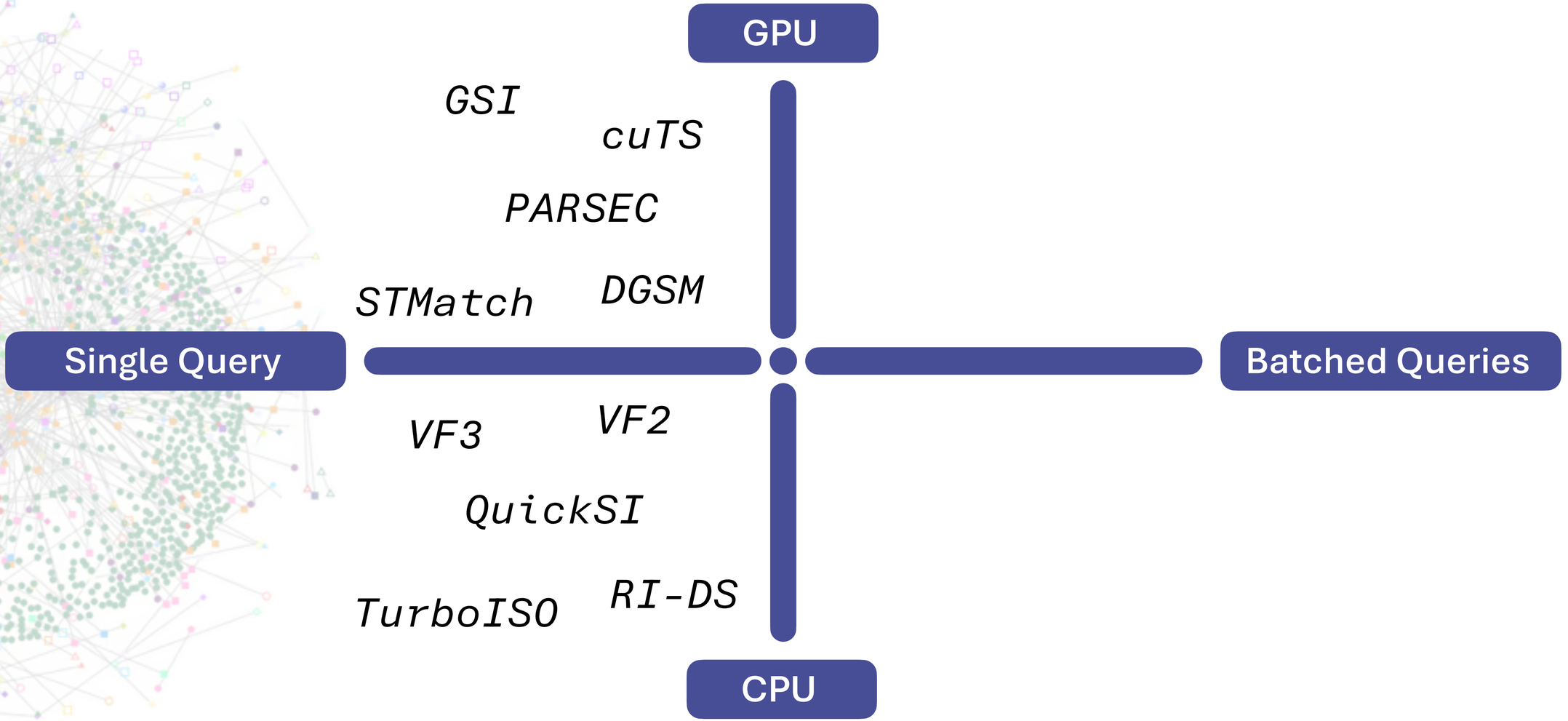
State-of-the-Art Landscape



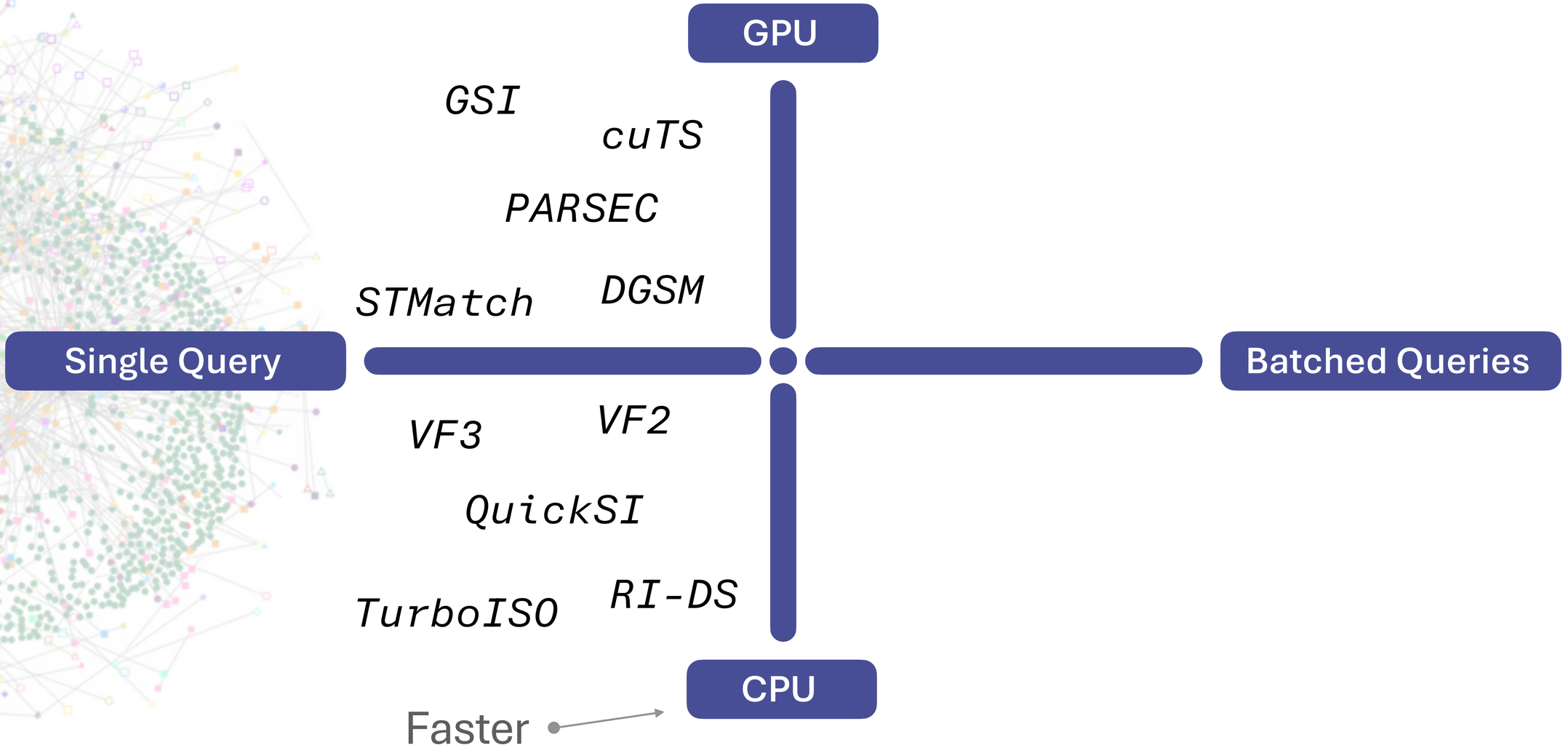
State-of-the-Art Landscape



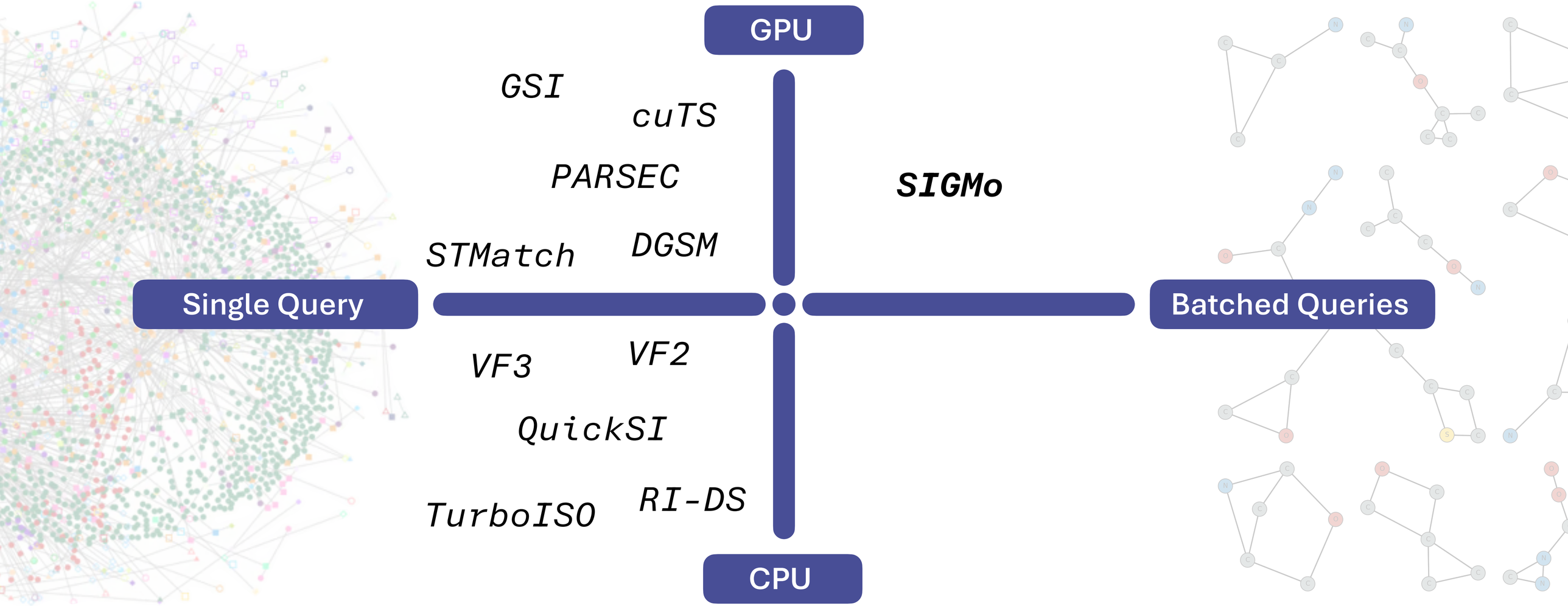
State-of-the-Art Landscape



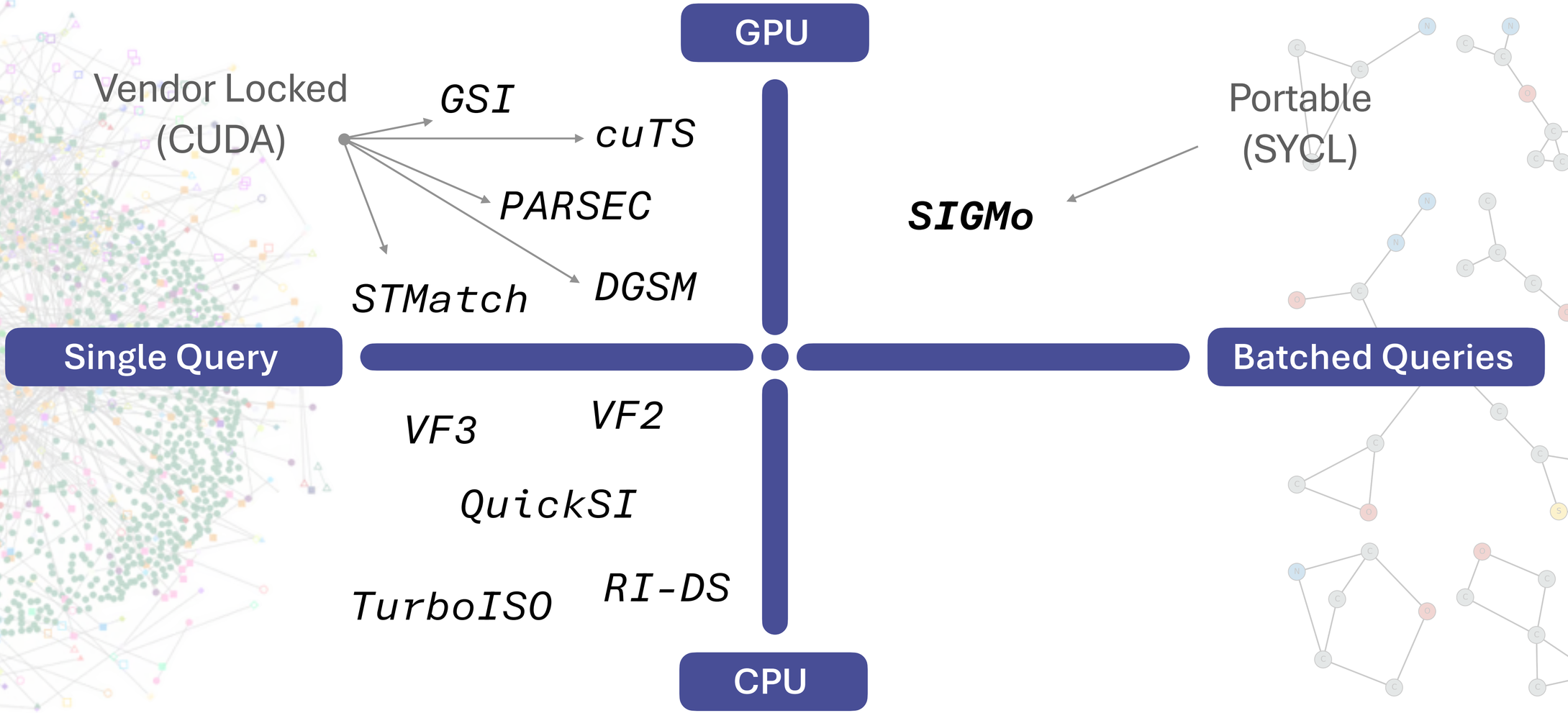
State-of-the-Art Landscape



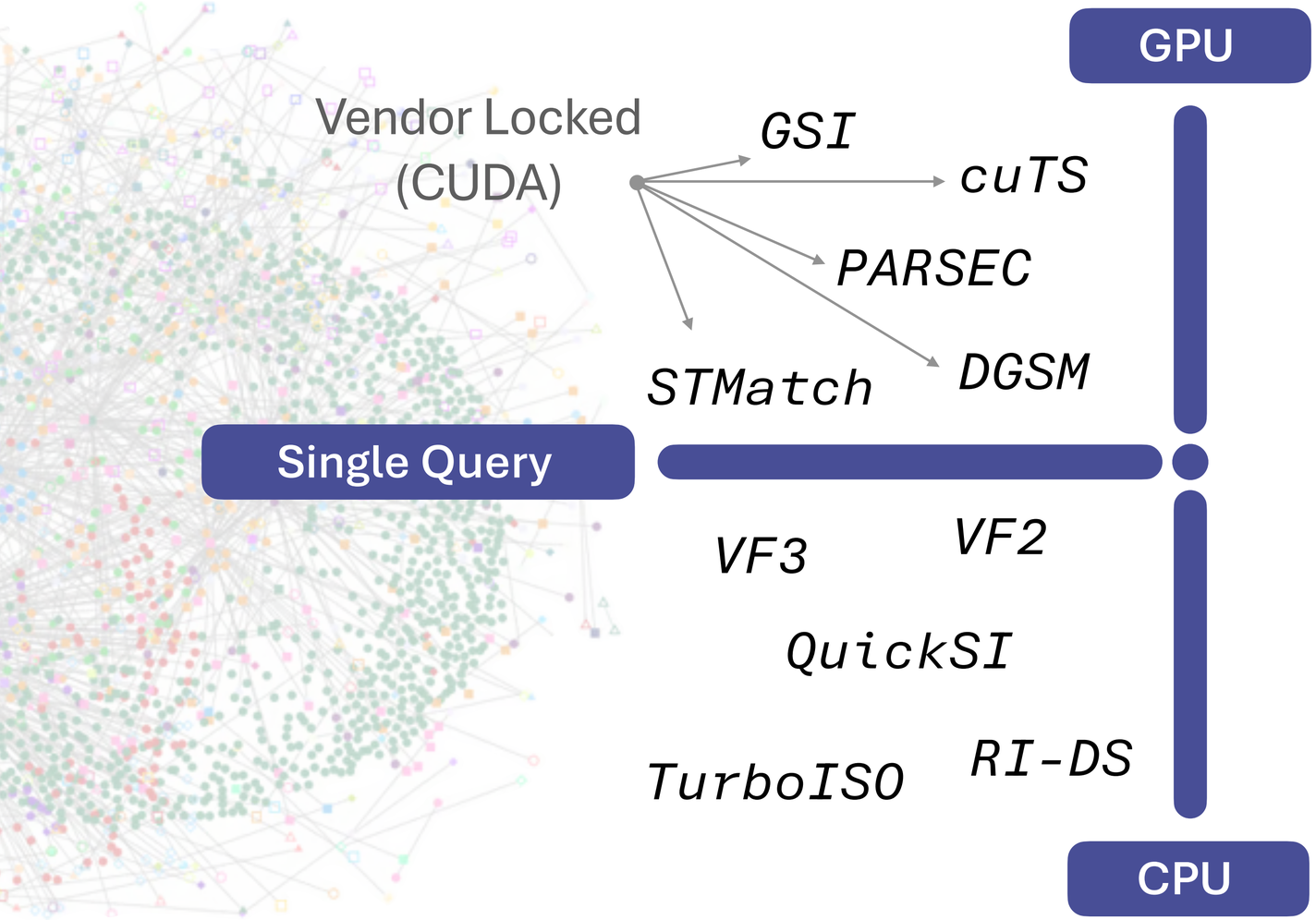
State-of-the-Art Landscape



State-of-the-Art Landscape



State-of-the-Art Landscape



SIGMo beats SOTA

33 × vs CPU

up to 1470 × vs GPU

Introducing SIGMo: a GPU-Batched, Domain-Aware SI Framework

*Designed specifically for **high-throughput** molecular matching*

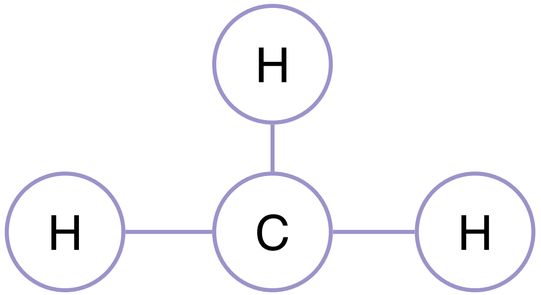
- Exploits **molecular constraints** (low degree, limited labels, …)
- Is based on a ***Filter-and-Join*** strategy, first described by Hullmann

Introducing SIGMo: a GPU-Batched, Domain-Aware SI Framework

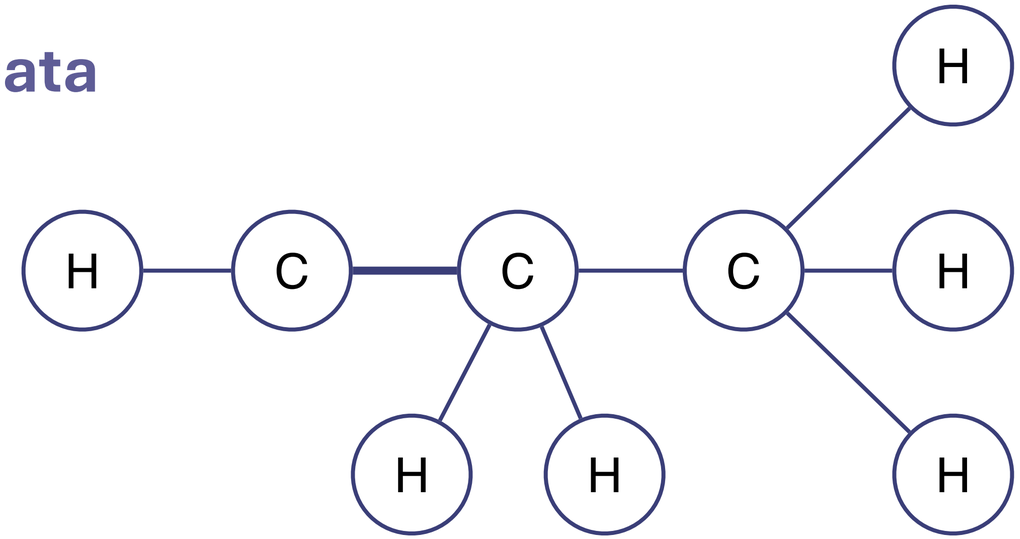
*Designed specifically for **high-throughput** molecular matching*

- Exploits **molecular constraints** (low degree, limited labels, …)
- Is based on a ***Filter-and-Join*** strategy, first described by Hullmann

Query



Data

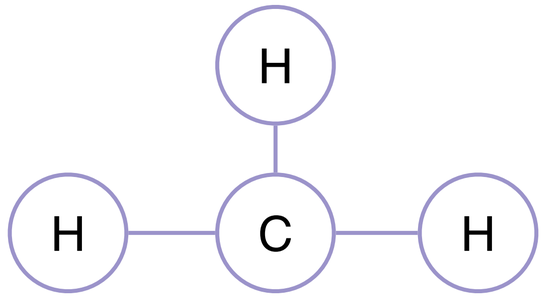


Introducing SIGMo: a GPU-Batched, Domain-Aware SI Framework

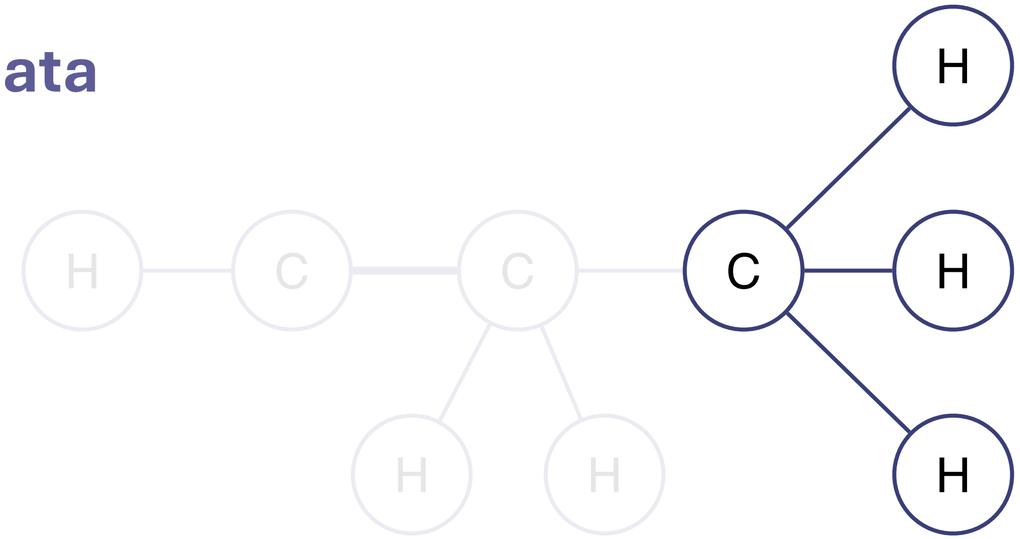
*Designed specifically for **high-throughput** molecular matching*

- Exploits **molecular constraints** (low degree, limited labels, …)
- Is based on a ***Filter-and-Join*** strategy, first described by Hullmann

Query



Data



Coming next

Coming next

Key Insight #1 Optimized for parallel batching of query & data graphs

Coming next

Key Insight #1 Optimized for parallel batching of query & data graphs

Key Insight #2 Aggressively prunes with iterative neighborhood signatures

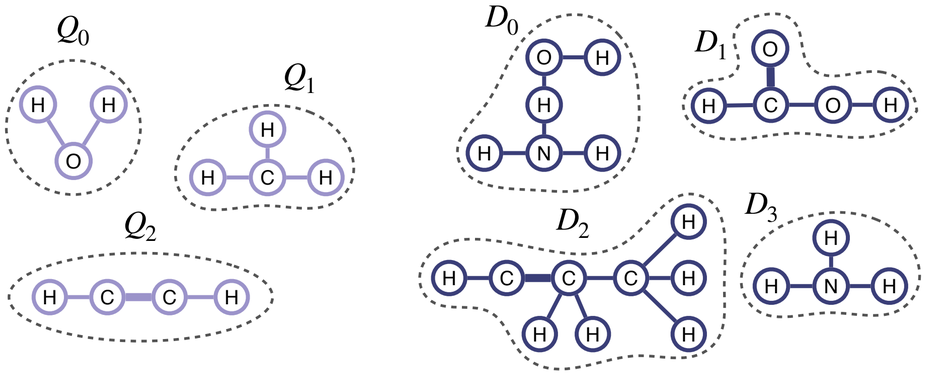
Coming next

Key Insight #1 Optimized for parallel batching of query & data graphs

Key Insight #2 Aggressively prunes with iterative neighborhood signatures

Key Insight #3 Joins candidates using a GPU-friendly stack-based DFS

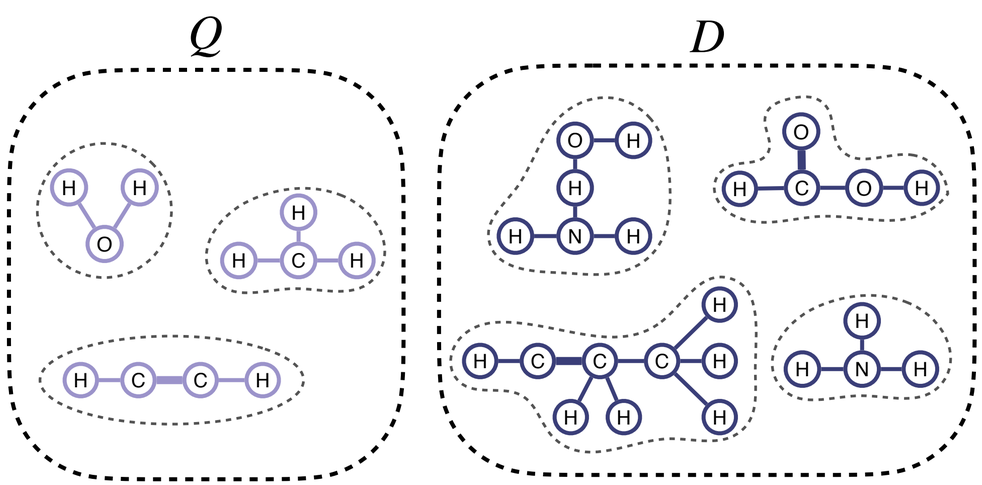
Key Insight #1: **Optimized for parallel batching on query & data graph**



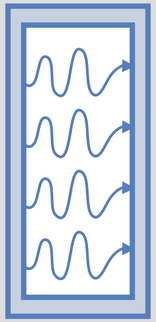
FILTER

JOIN

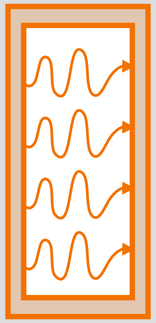
Key Insight #1: **Optimized for parallel batching on query & data graph**



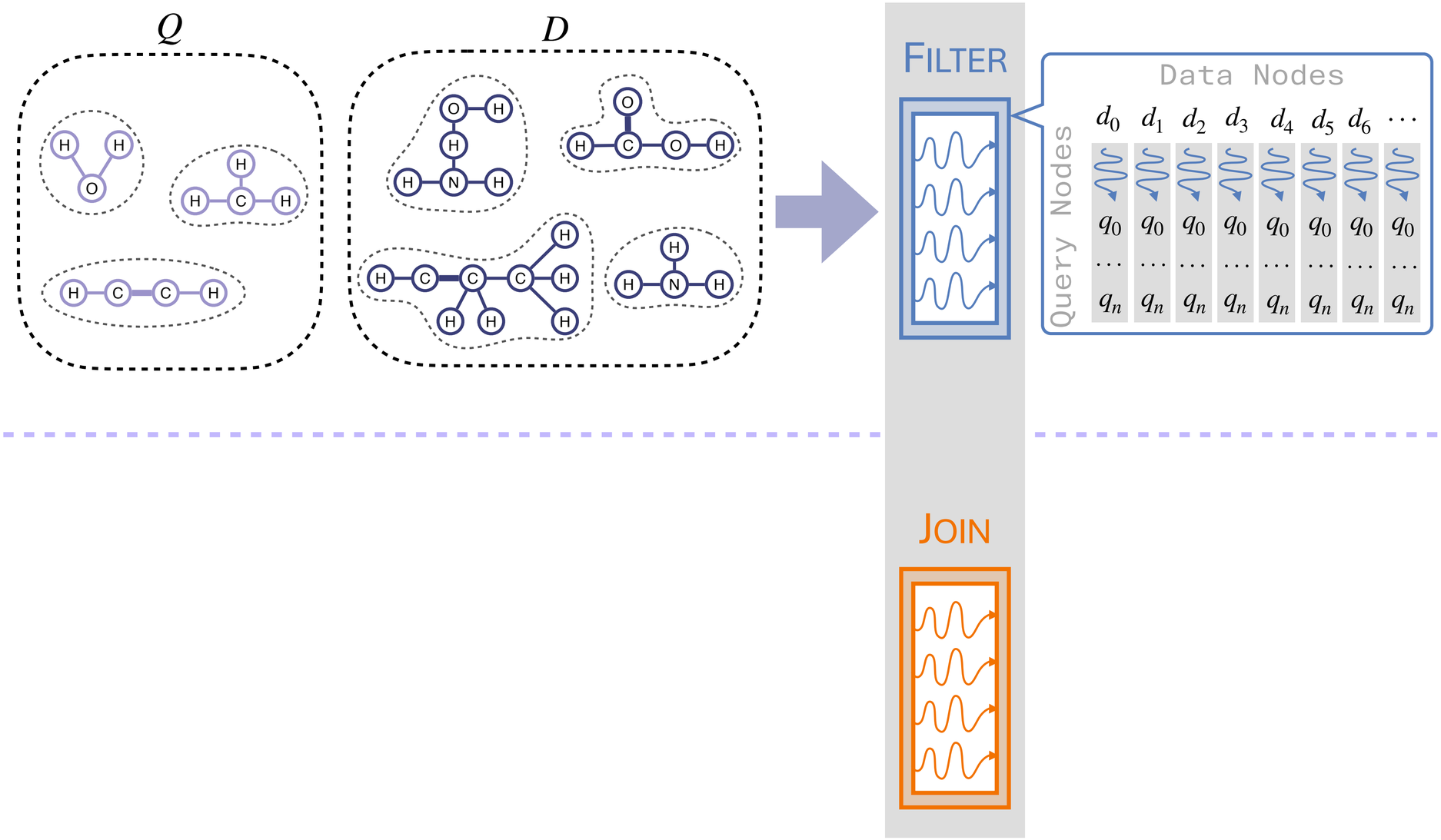
FILTER



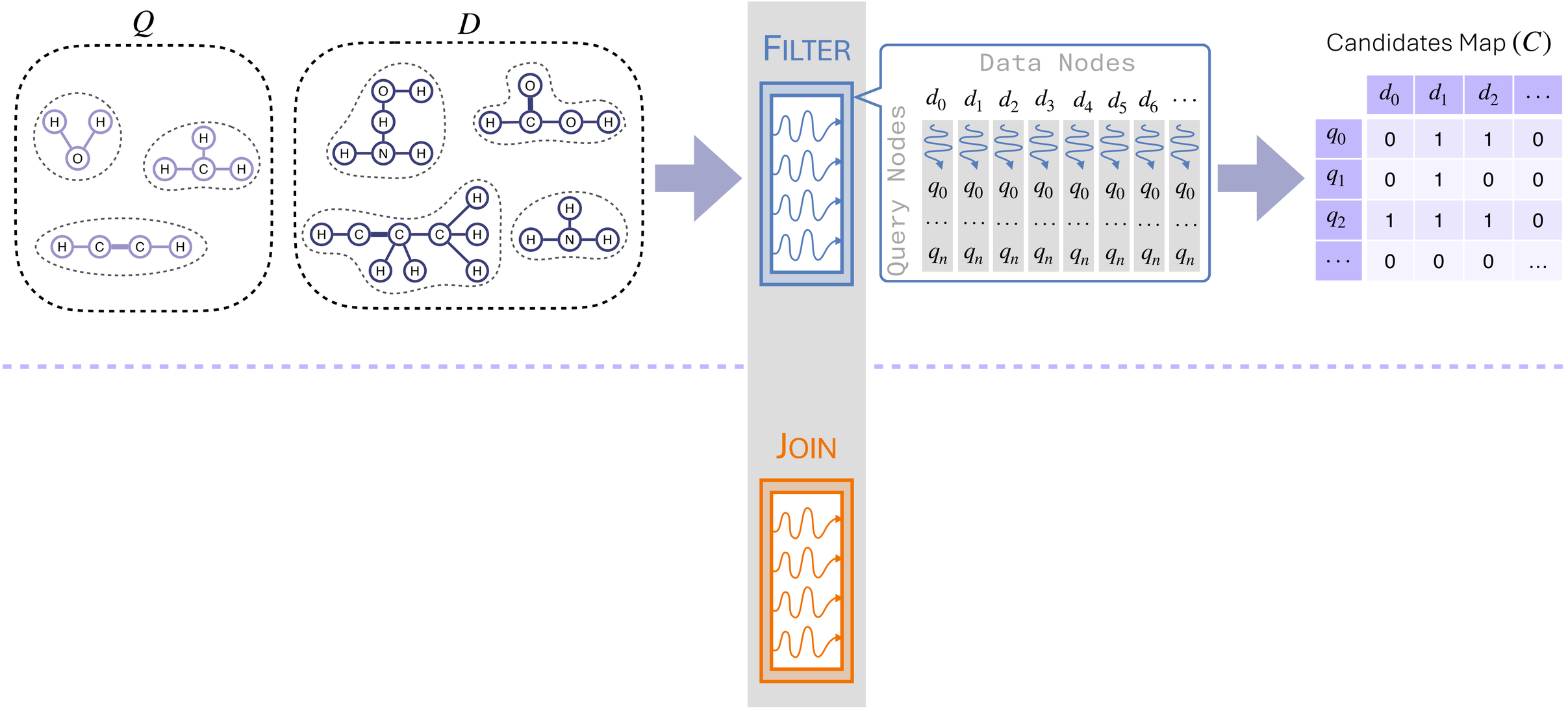
JOIN



Key Insight #1: **Optimized for parallel batching on query & data graph**



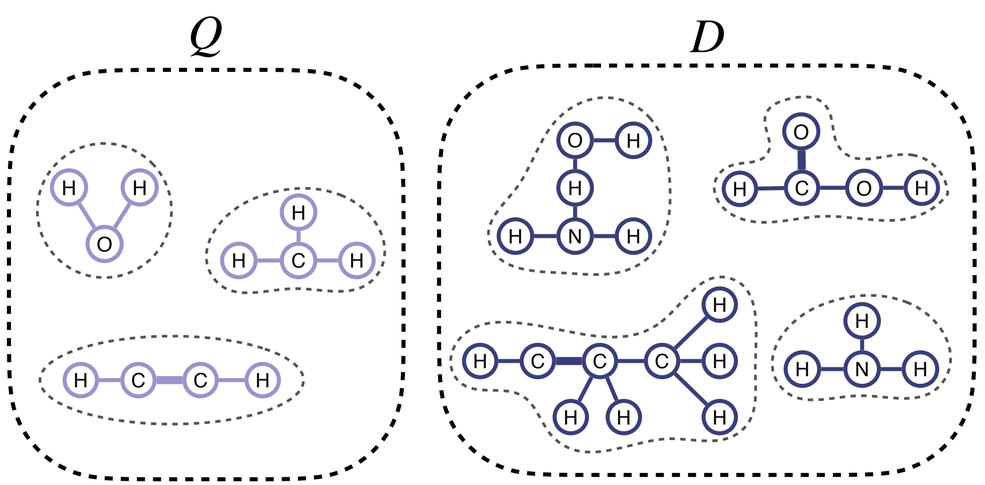
Key Insight #1: **Optimized for parallel batching on query & data graph**



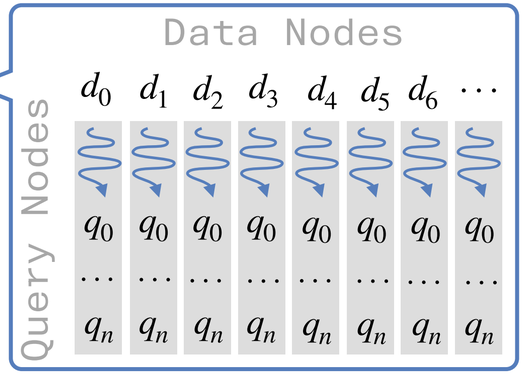
Candidates Map (C)

	d_0	d_1	d_2	⋯
q_0	0	1	1	0
q_1	0	1	0	0
q_2	1	1	1	0
⋯	0	0	0	…

Key Insight #1: **Optimized for parallel batching on query & data graph**

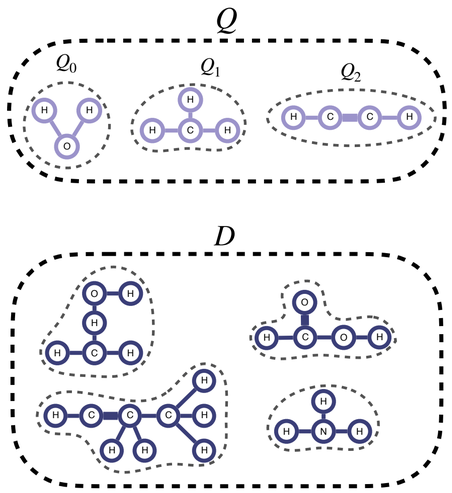


FILTER



Candidates Map (C)

	d_0	d_1	d_2	$\cdots$
q_0	0	1	1	0
q_1	0	1	0	0
q_2	1	1	1	0
$\cdots$	0	0	0	…

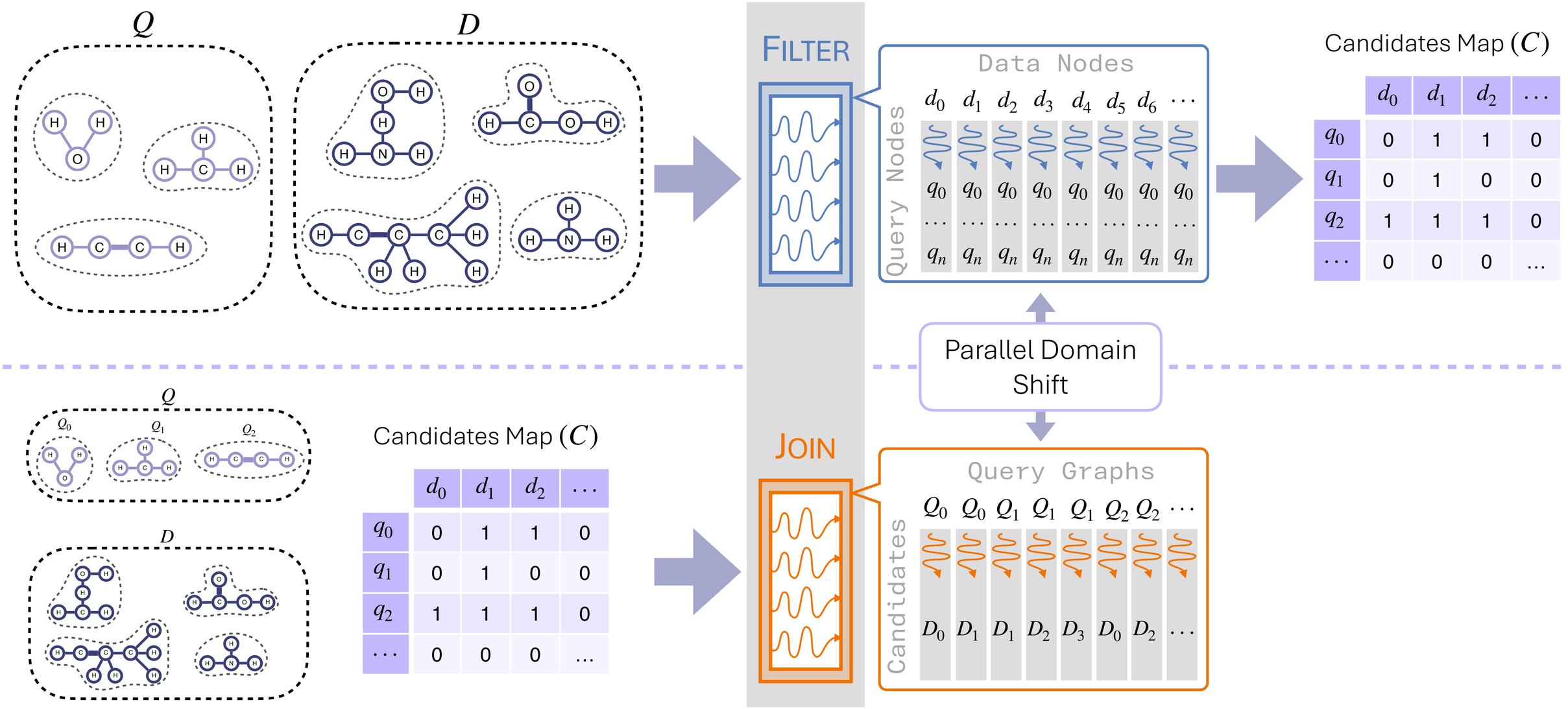


Candidates Map (C)

	d_0	d_1	d_2	$\cdots$
q_0	0	1	1	0
q_1	0	1	0	0
q_2	1	1	1	0
$\cdots$	0	0	0	…

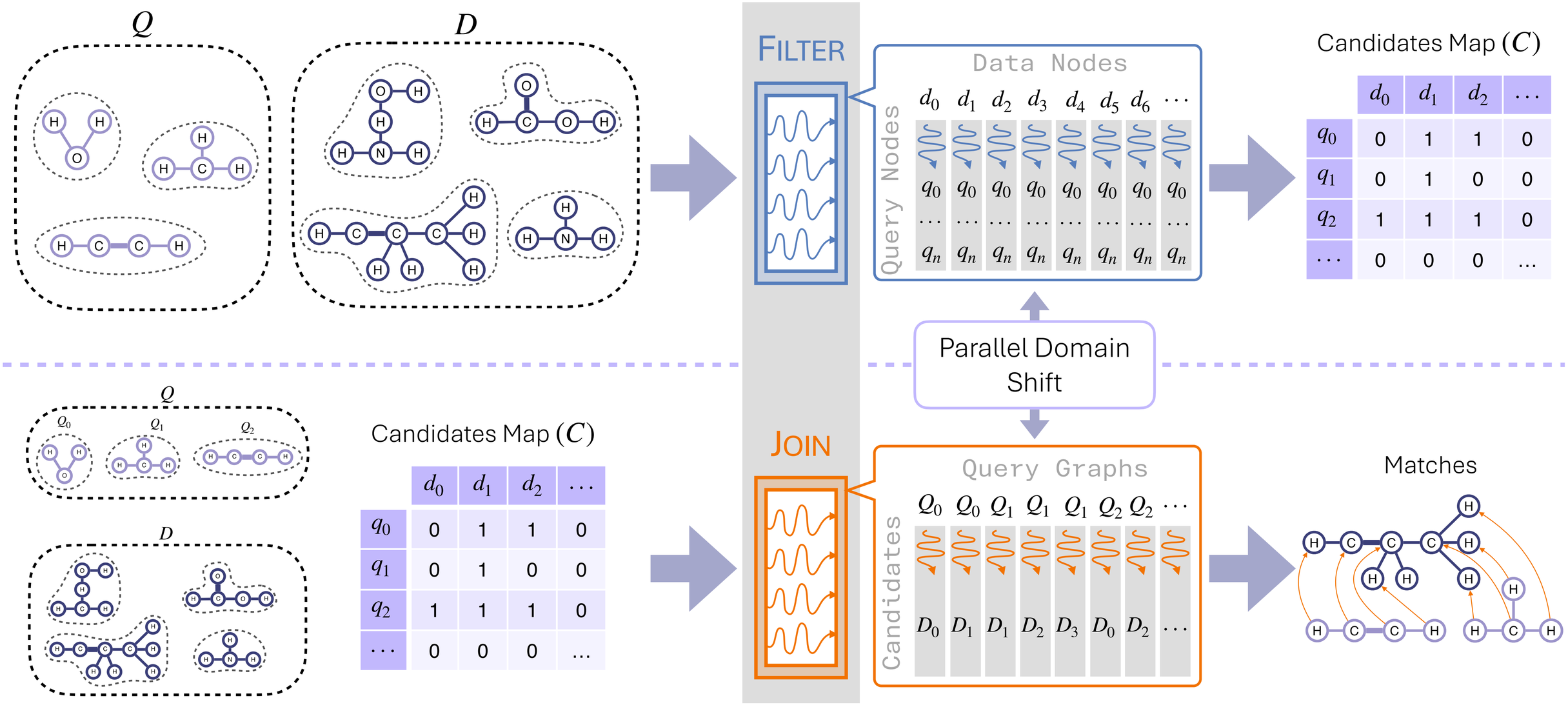
JOIN

Key Insight #1: **Optimized for parallel batching on query & data graph**



	d_0	d_1	d_2	$\cdots$
q_0	0	1	1	0
q_1	0	1	0	0
q_2	1	1	1	0
$\cdots$	0	0	0	…

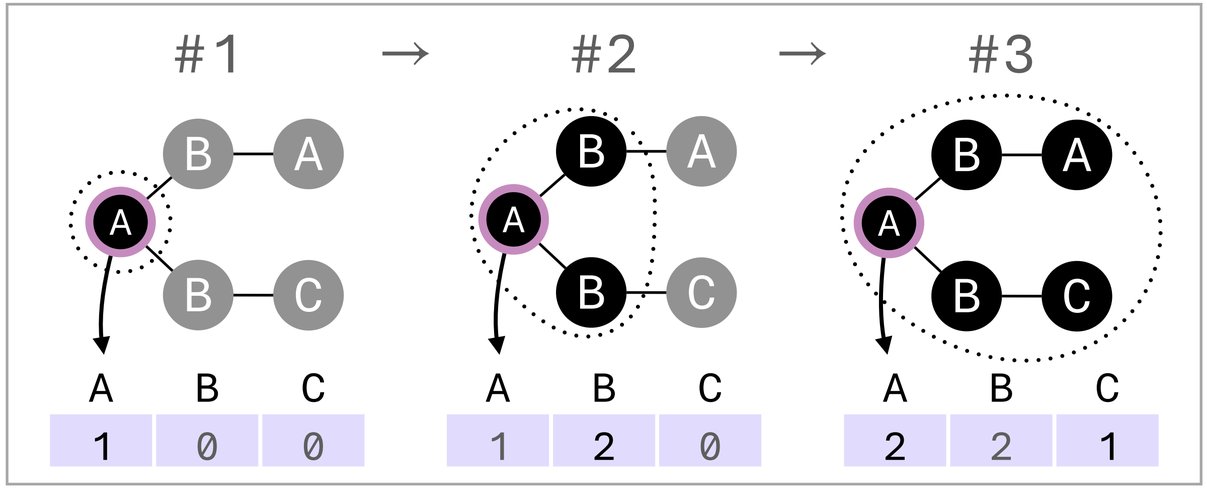
Key Insight #1: **Optimized for parallel batching on query & data graph**



Key Insight #2: **Iterative Neighborhood-Signature Filtering**

Key Insight #2: **Iterative Neighborhood-Signature Filtering**

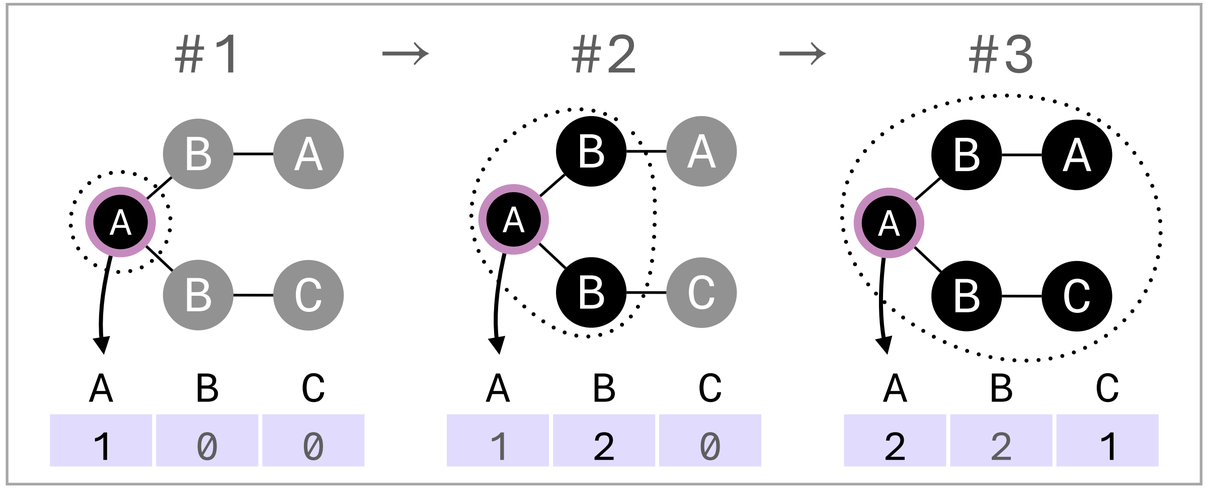
- Uses **nodes signatures** to prune candidates
  - a compact summary of the **occurrence of neighbor node labels** within its local neighborhood



Signature refinement of the marked node

Key Insight #2: **Iterative Neighborhood-Signature Filtering**

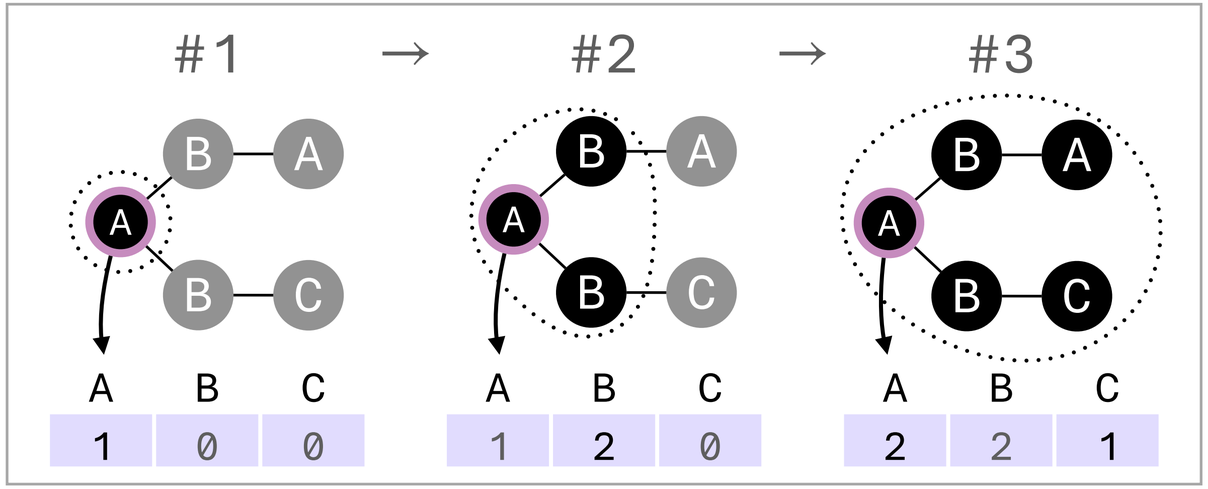
- Uses **nodes signatures** to prune candidates
  - a compact summary of the **occurrence of neighbor node labels** within its local neighborhood
- Signatures are refined iteratively by **enlarging the neighborhood** context at each step



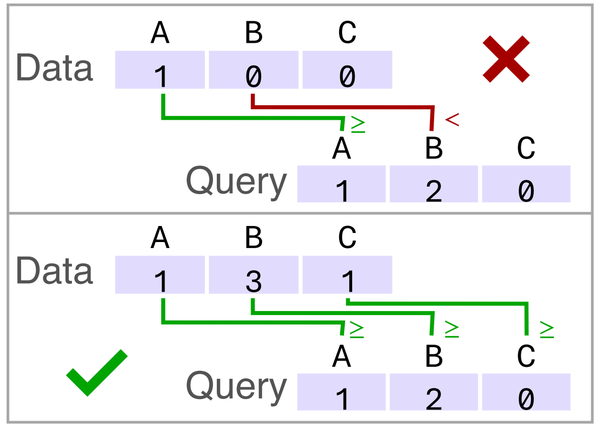
Signature refinement of the marked node

Key Insight #2: **Iterative Neighborhood-Signature Filtering**

- Uses **nodes signatures** to prune candidates
  - a compact summary of the **occurrence of neighbor node labels** within its local neighborhood
- Signatures are refined iteratively by **enlarging the neighborhood** context at each step
- A data candidate is discarded when its signature cannot "**dominate**" the query's signature



Signature refinement of the marked node

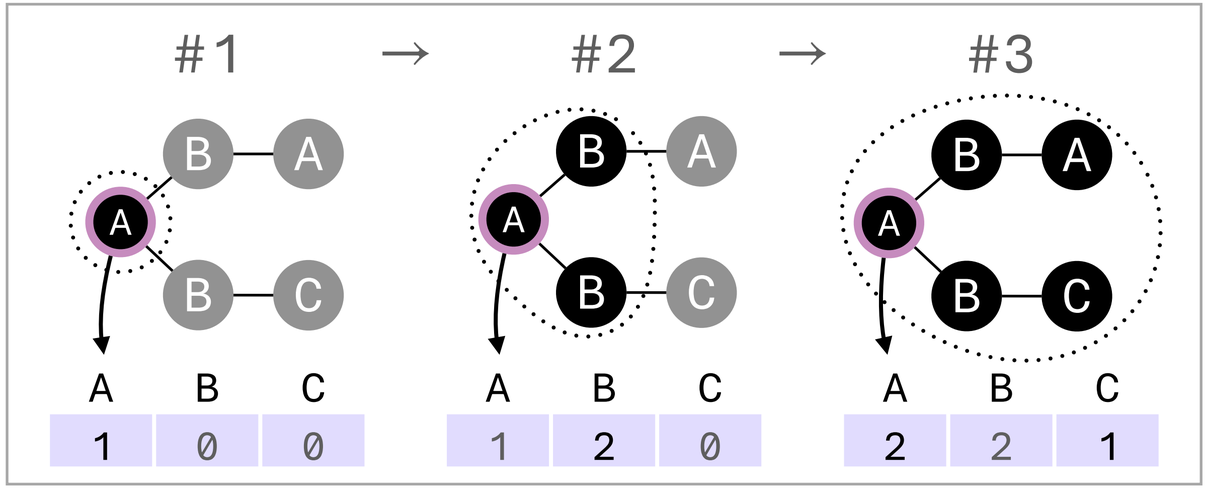


Candidates Pruning

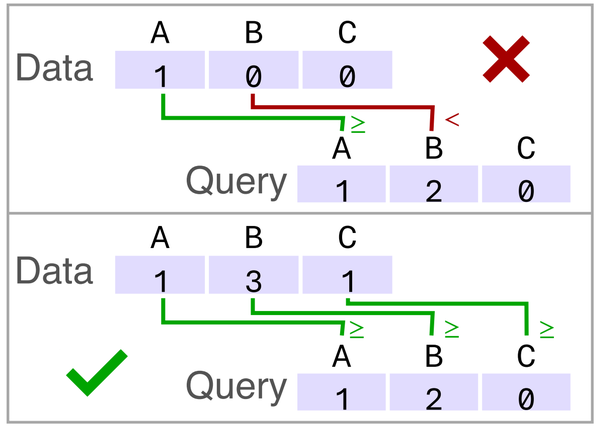
Key Insight #2: **Iterative Neighborhood-Signature Filtering**

- Uses **nodes signatures** to prune candidates
  - a compact summary of the **occurrence of neighbor node labels** within its local neighborhood
- Signatures are refined iteratively by **enlarging the neighborhood** context at each step
- A data candidate is discarded when its signature cannot "**dominate**" the query's signature

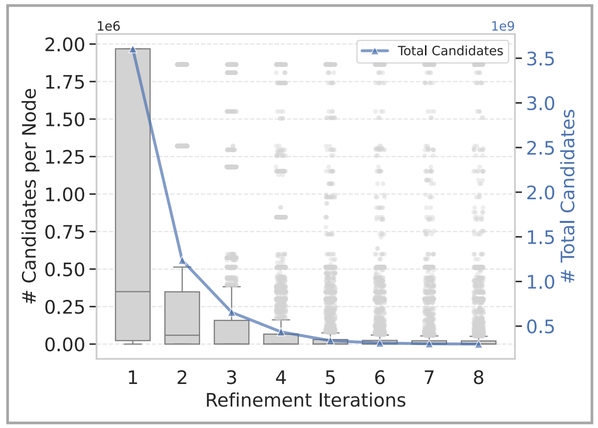
➡ After a few iterations, most invalid mappings are eliminated before the join phase



Signature refinement of the marked node

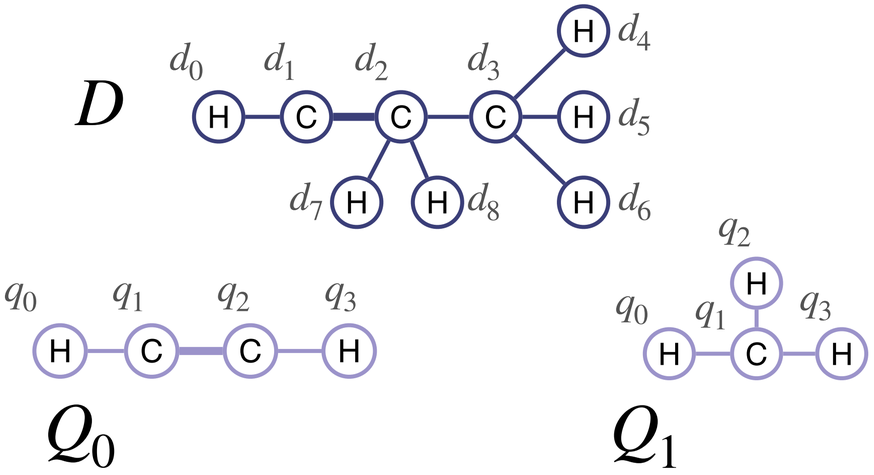


Candidates Pruning



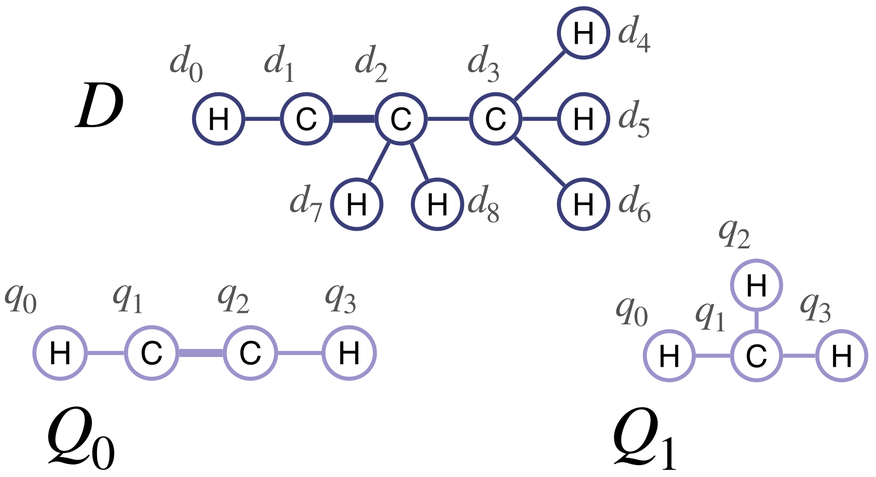
Candidates evolution over iterations

Key Insight #3: **Joins candidates using a GPU-friendly stack-based DFS**



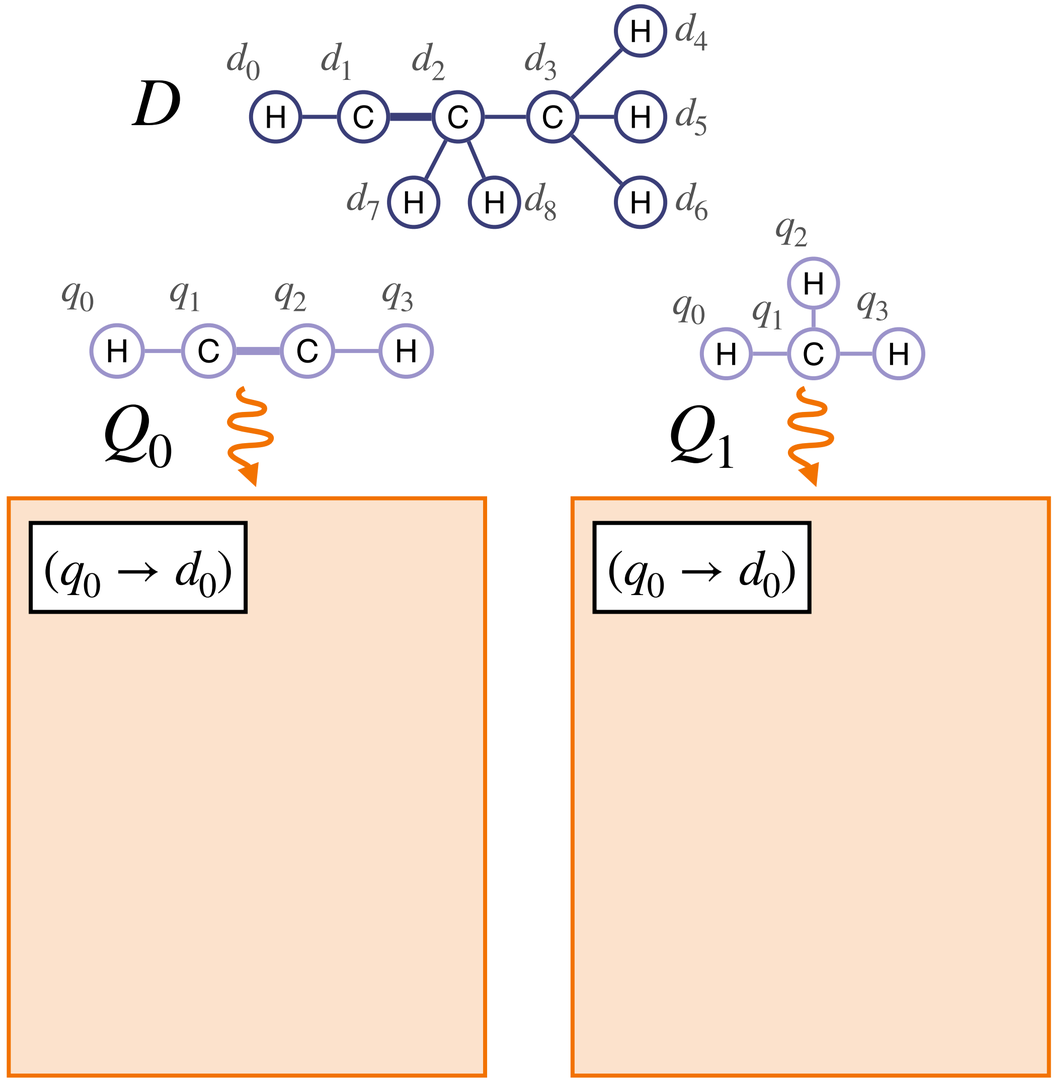
Key Insight #3: **Joins candidates using a GPU-friendly stack-based DFS**

- Starts by **constructing valid query & data** graph pairs according to the **candidate map**



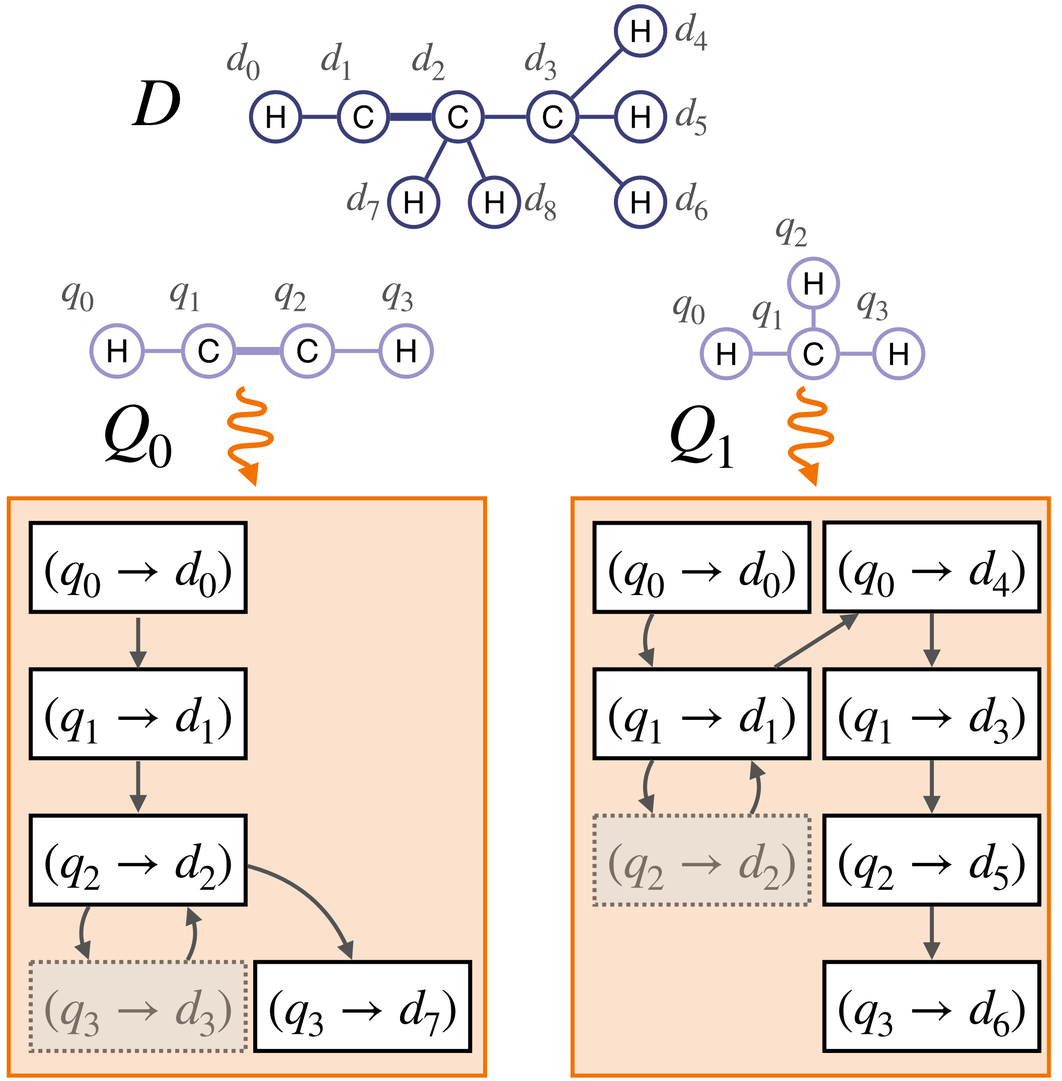
Key Insight #3: **Joins candidates using a GPU-friendly stack-based DFS**

- Starts by **constructing valid query & data** graph pairs according to the **candidate map**
- Iterative, **non-recursive** DFS
  - per thread **private stack** explores possible mappings



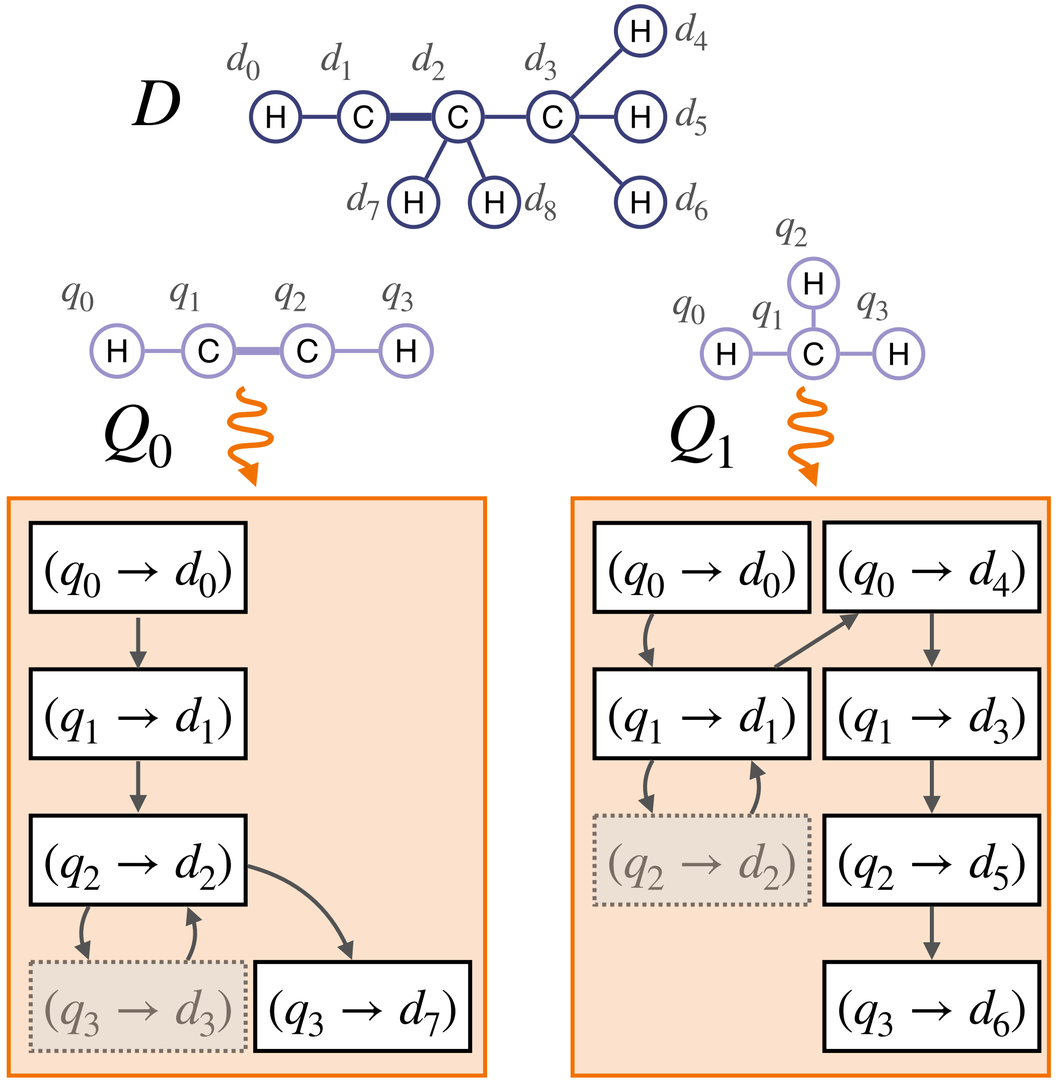
Key Insight #3: **Joins candidates using a GPU-friendly stack-based DFS**

- Starts by **constructing valid query & data** graph pairs according to the **candidate map**
- Iterative, **non-recursive** DFS
  - per thread **private stack** explores possible mappings



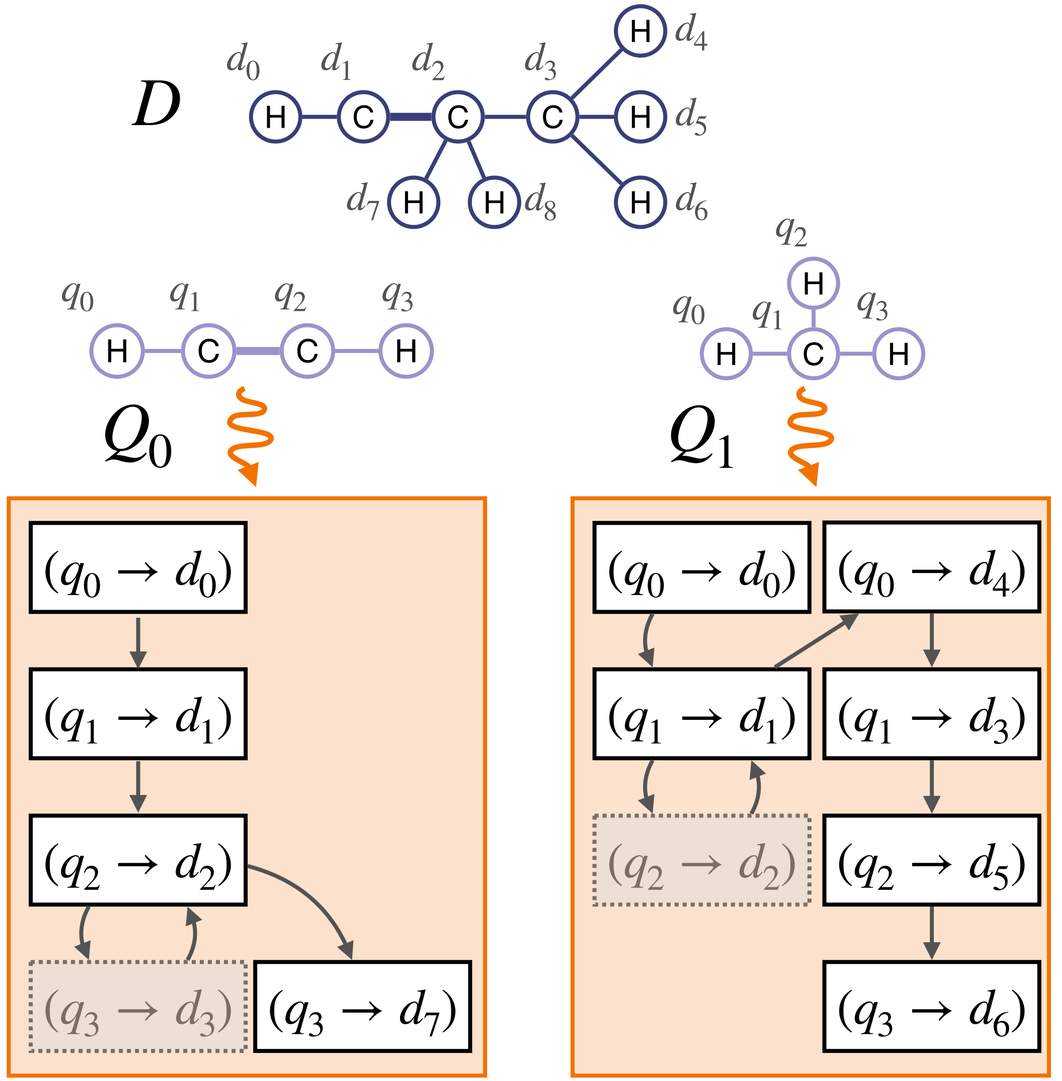
Key Insight #3: **Joins candidates using a GPU-friendly stack-based DFS**

- Starts by **constructing valid query & data** graph pairs according to the **candidate map**
- Iterative, **non-recursive** DFS
  - per thread **private stack** explores possible mappings
- Small, **bounded depth** up to the size of the query graph (≈ 30)



Key Insight #3: **Joins candidates using a GPU-friendly stack-based DFS**

- Starts by **constructing valid query & data** graph pairs according to the **candidate map**
- Iterative, **non-recursive** DFS
  - per thread **private stack** explores possible mappings
- Small, **bounded depth** up to the size of the query graph (≈ 30)
- For each query & data pair supports two execution models:
  - ***Find-First*** stops at first pair match
  - ***Find-All*** finds all matches in a pair



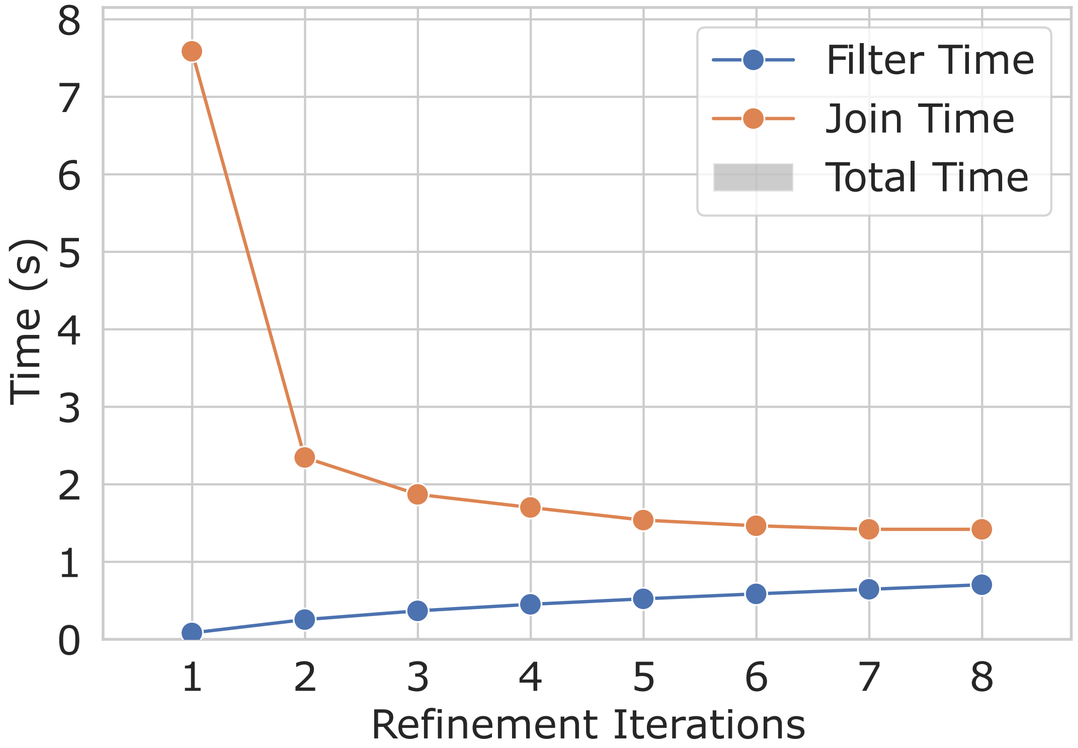
Experimental Evaluation – **Assessing SIGMo on NVIDIA V100S**

Experimental Evaluation – **Assessing SIGMo on NVIDIA V100S**

- **Dataset**: *618* query graphs (*3,413 nodes*) and *114,901* data graphs (*2,745,872 nodes*)

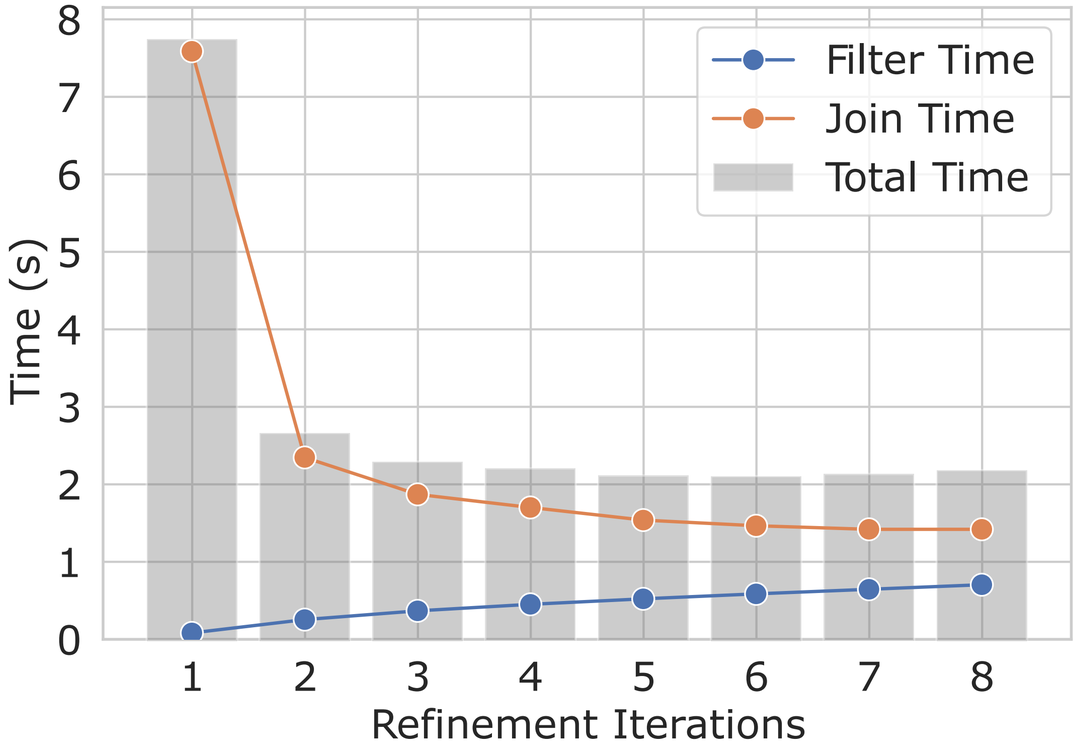
Experimental Evaluation – **Assessing SIGMo on NVIDIA V100S**

- **Dataset**: *618* query graphs (*3,413 nodes*) and *114,901* data graphs (*2,745,872 nodes*)



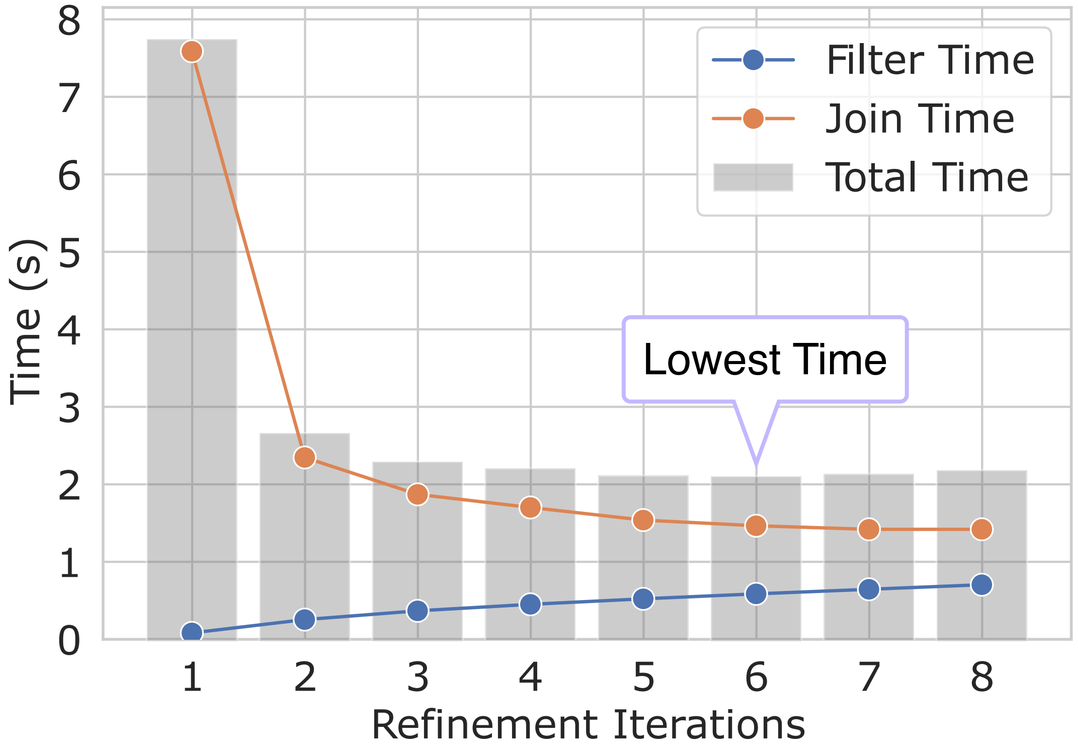
Experimental Evaluation – **Assessing SIGMo on NVIDIA V100S**

- **Dataset**: *618* query graphs (*3,413 nodes*) and *114,901* data graphs (*2,745,872 nodes*)



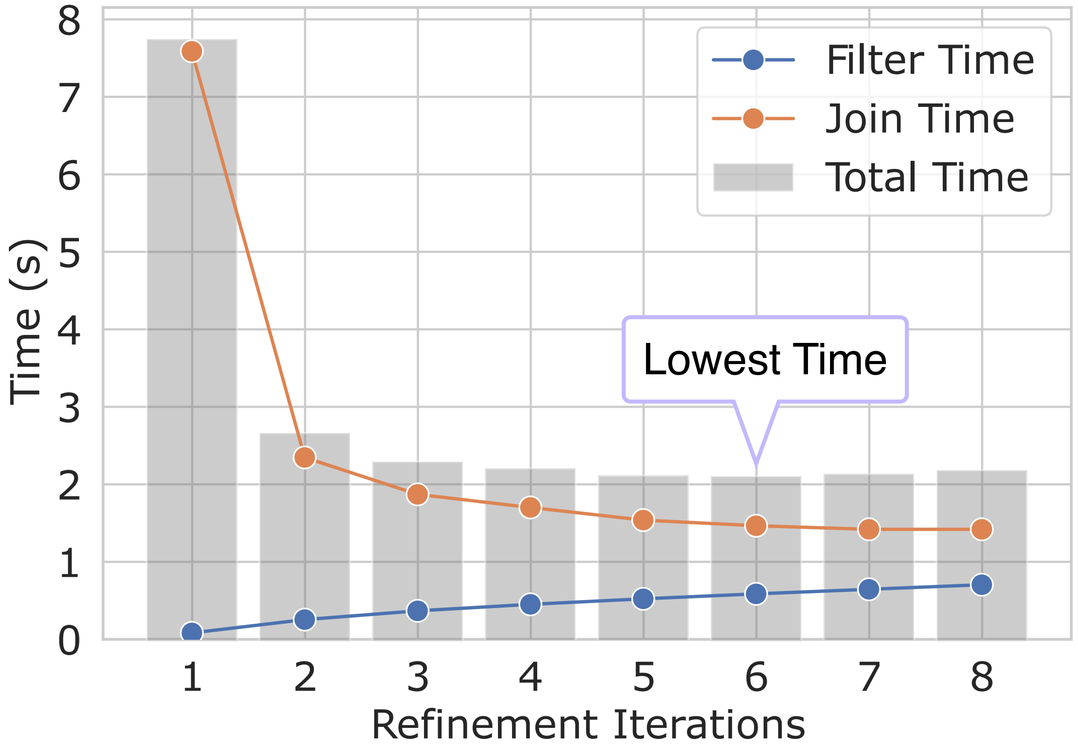
Experimental Evaluation – **Assessing SIGMo on NVIDIA V100S**

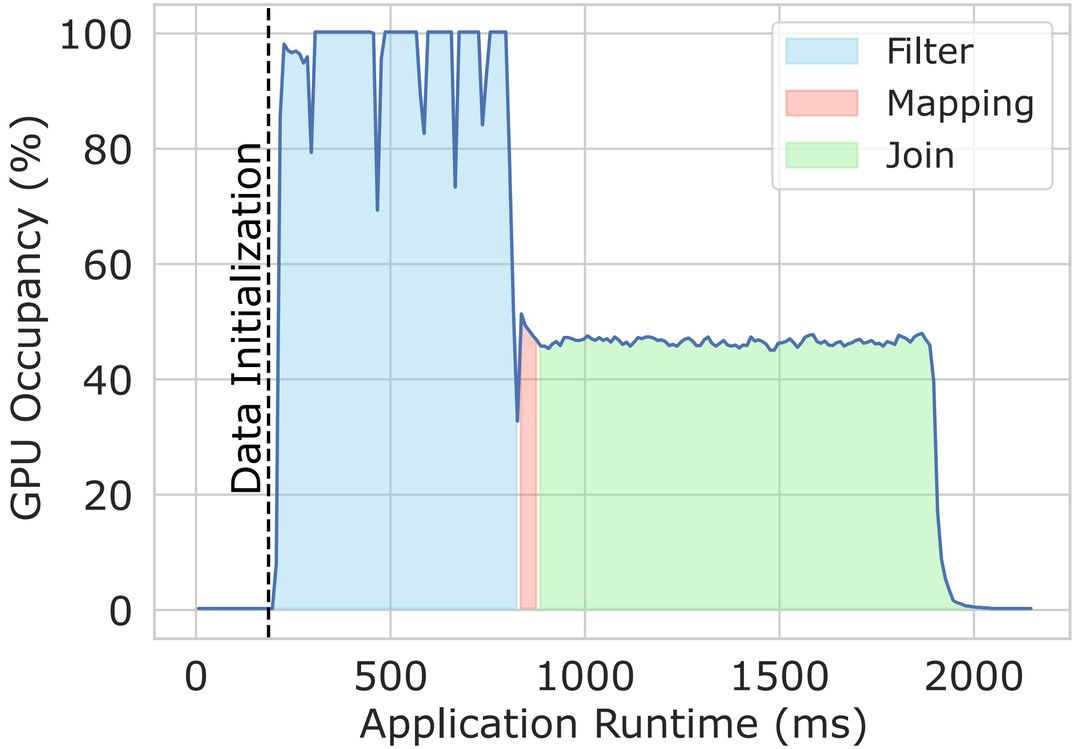
- **Dataset**: *618* query graphs (*3,413 nodes*) and *114,901* data graphs (*2,745,872 nodes*)



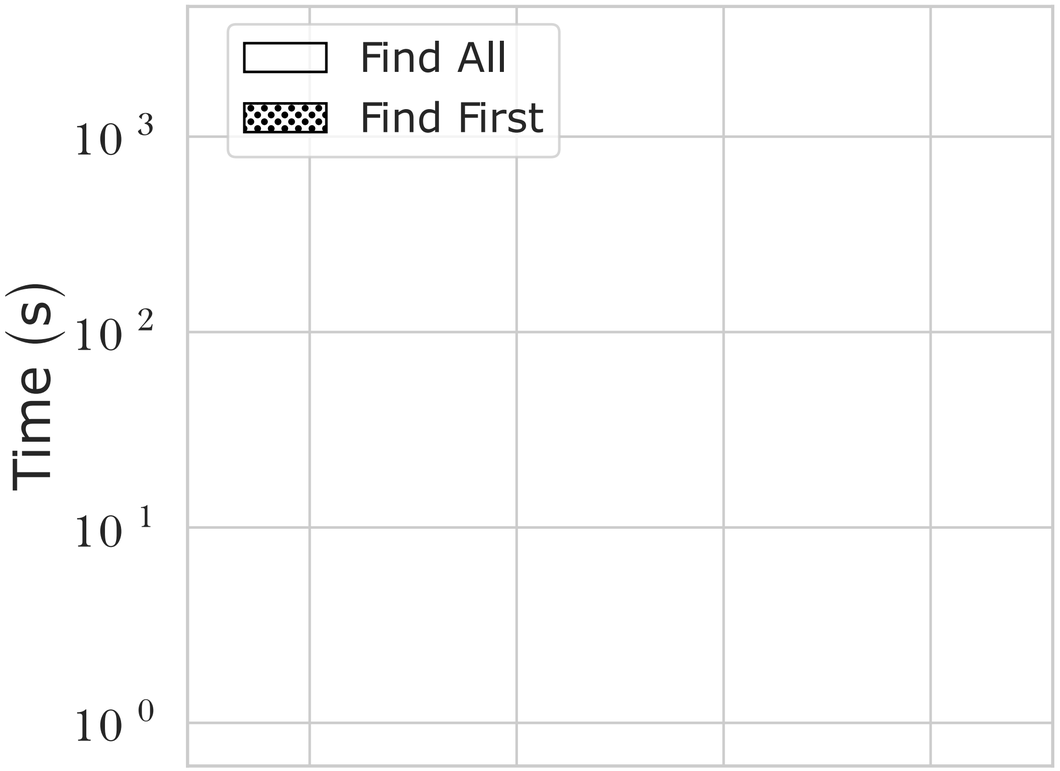
Experimental Evaluation – **Assessing SIGMo on NVIDIA V100S**

- **Dataset**: *618* query graphs (*3,413 nodes*) and *114,901* data graphs (*2,745,872 nodes*)

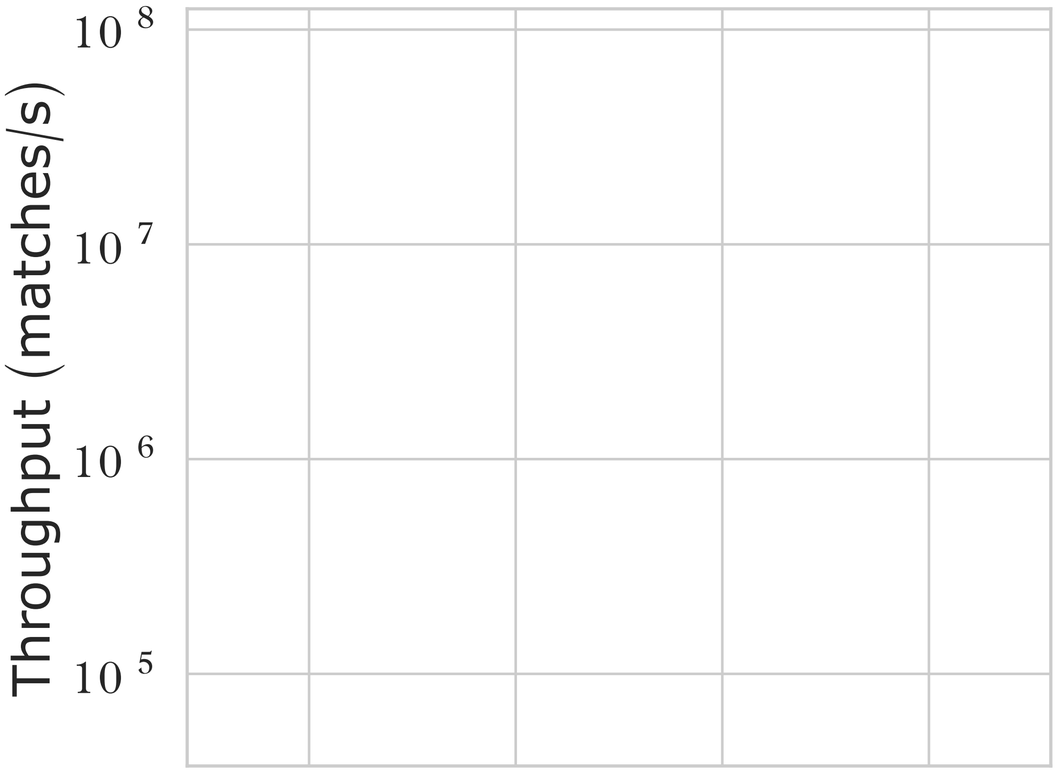




Experimental Evaluation – **State-of-the-Art Comparison**



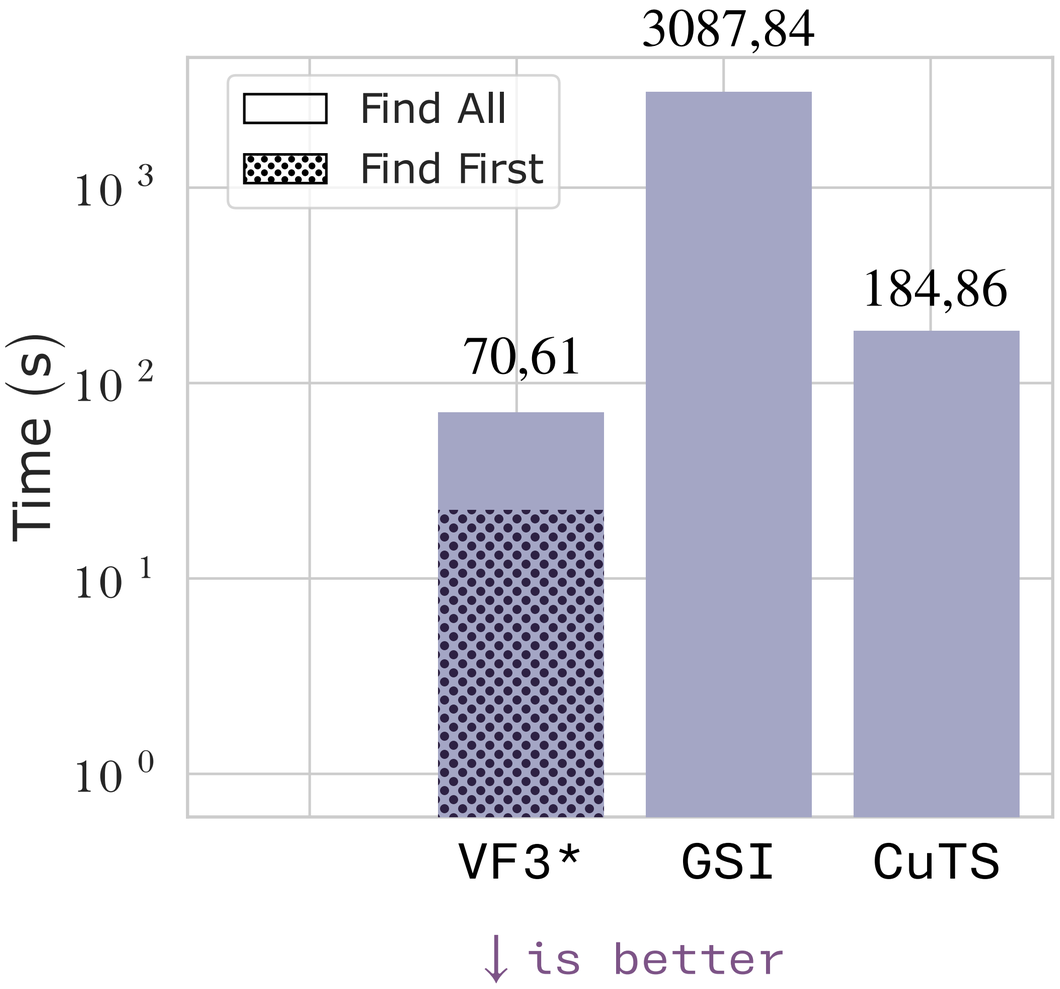
↓ is better

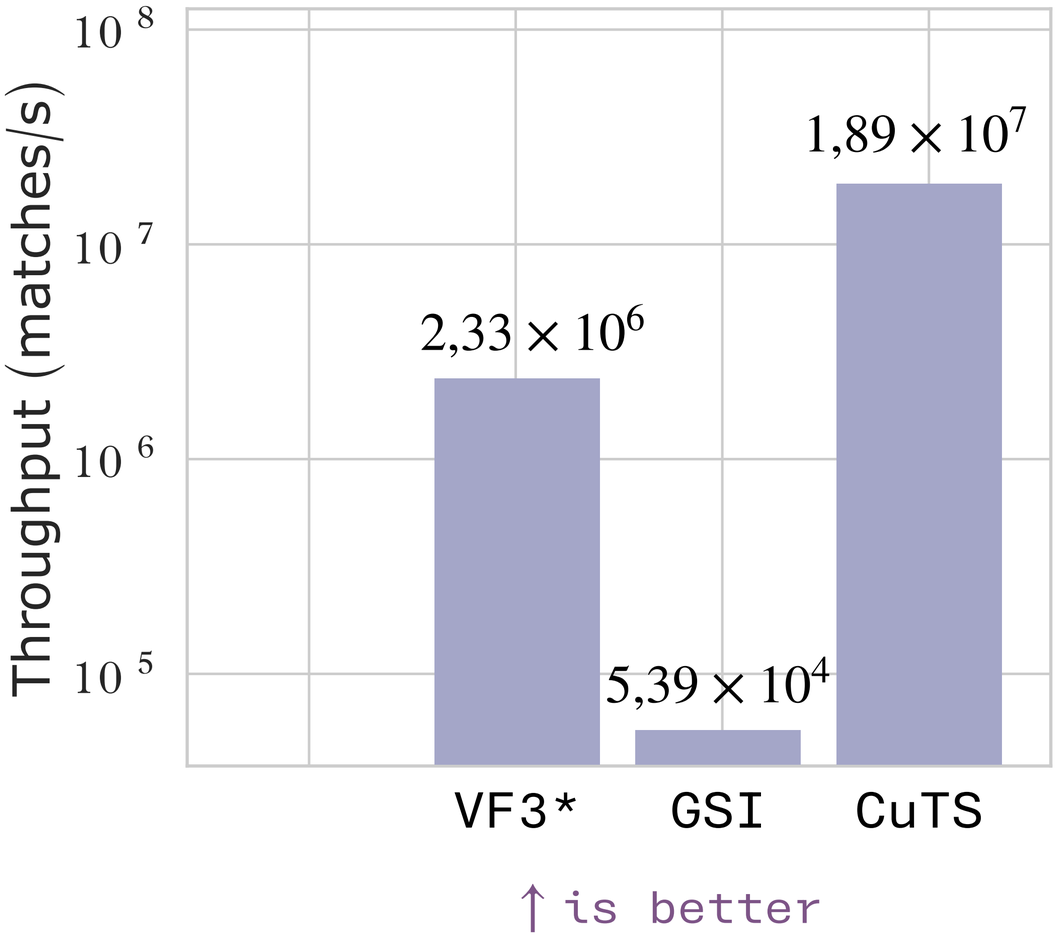


↑ is better

** runs on CPU*

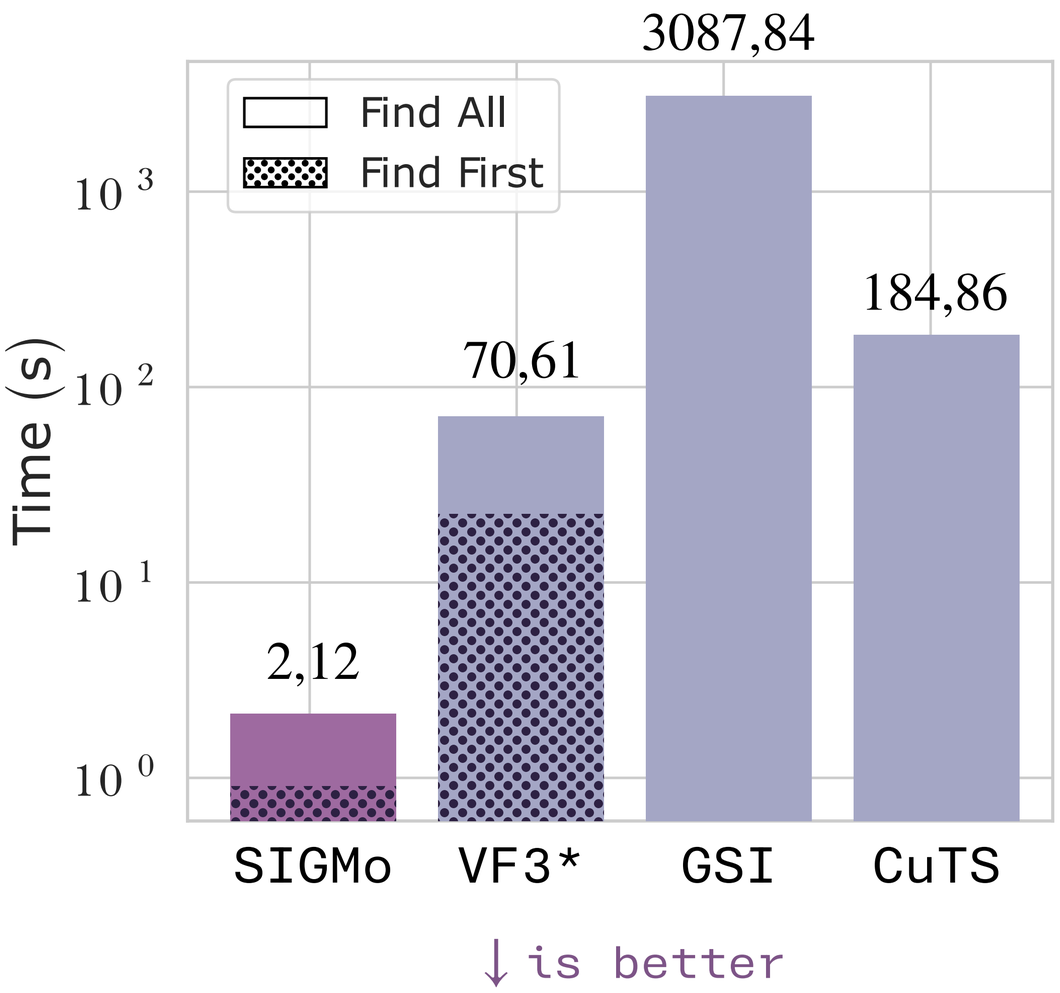
Experimental Evaluation – **State-of-the-Art Comparison**

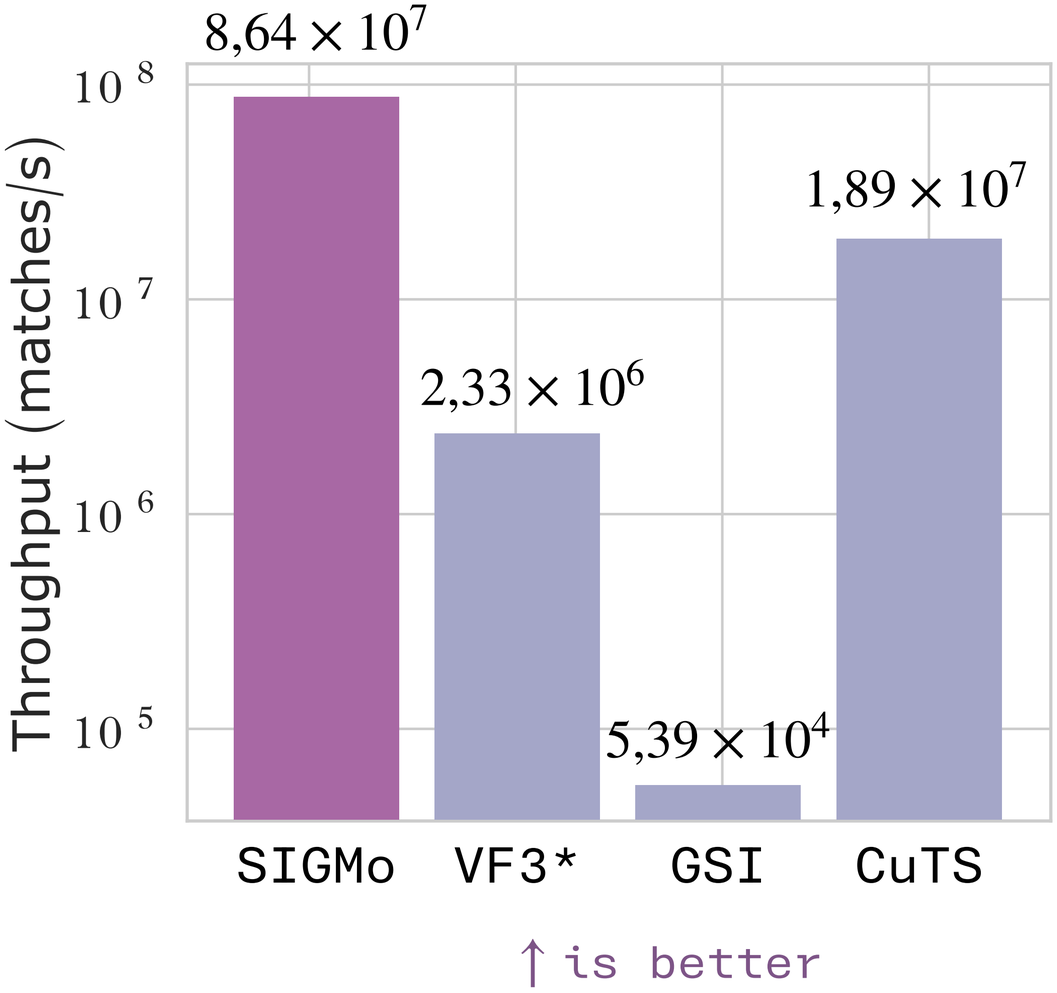




** runs on CPU*

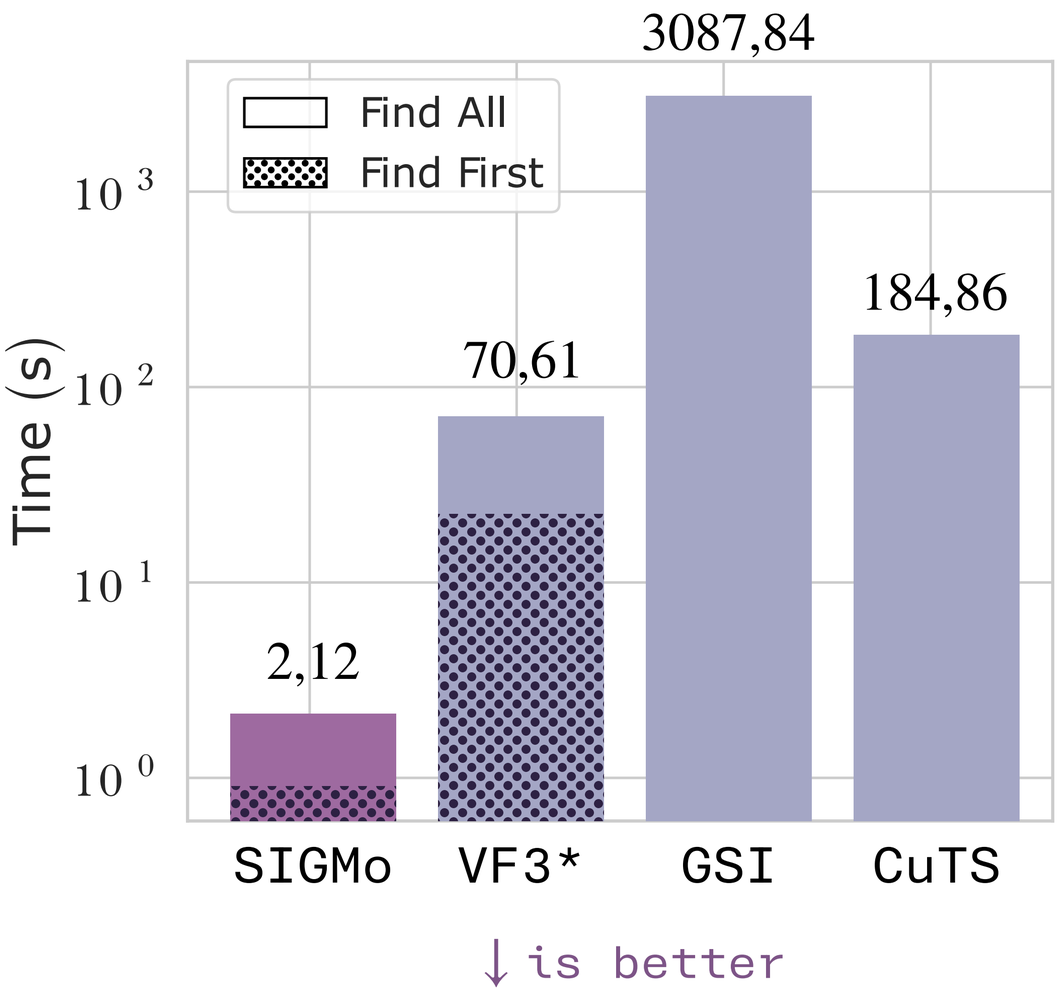
Experimental Evaluation – **State-of-the-Art Comparison**

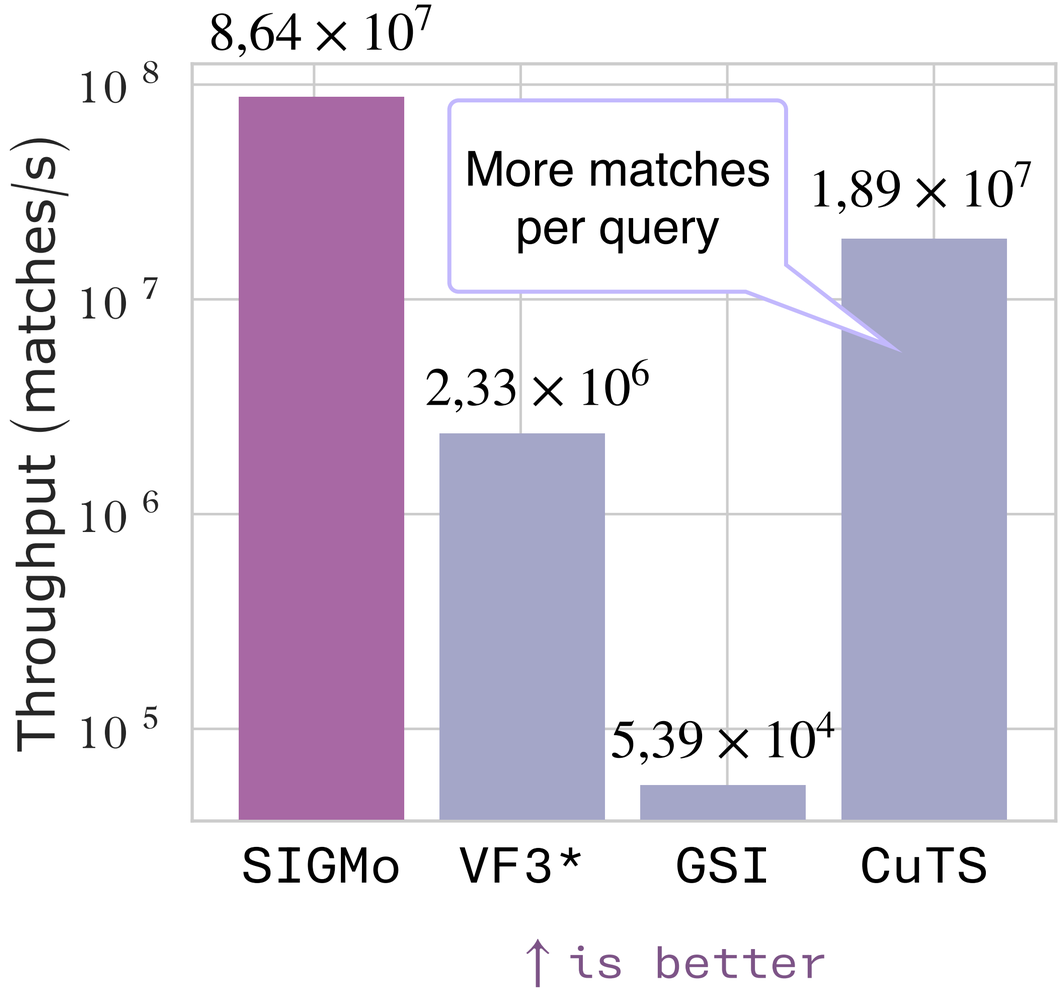




* runs on CPU

Experimental Evaluation – **State-of-the-Art Comparison**



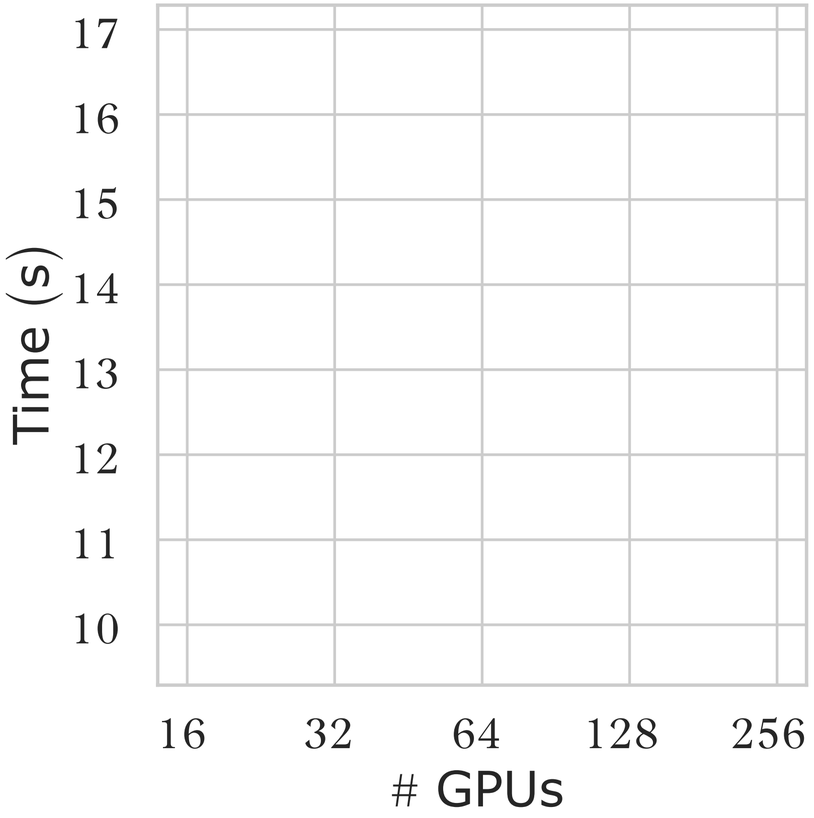


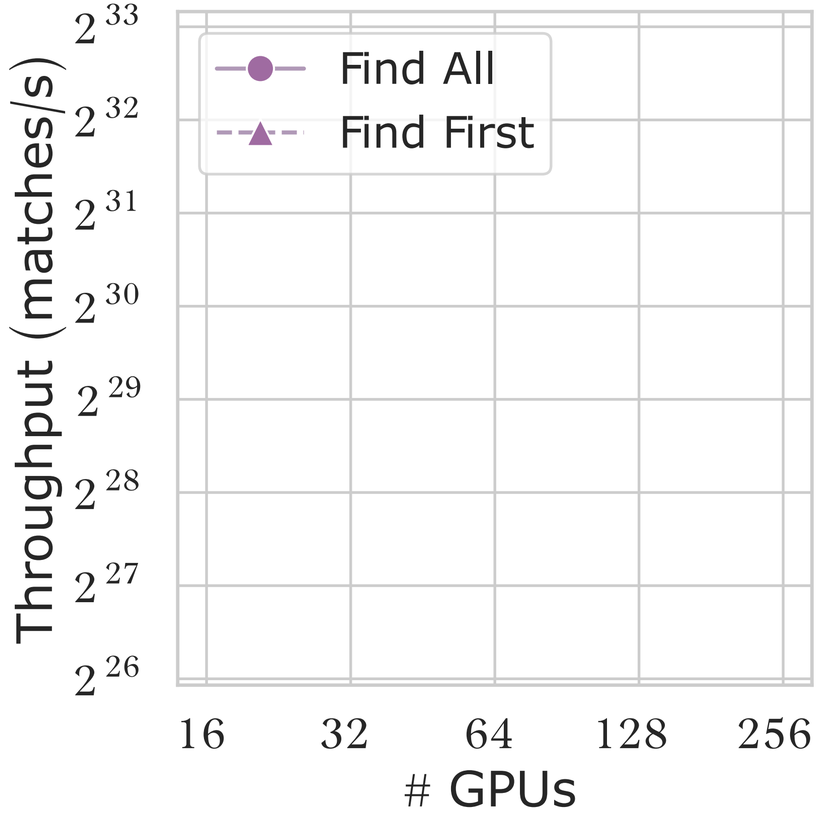
* *runs on CPU*

Experimental Evaluation – **Scalability Evaluation**

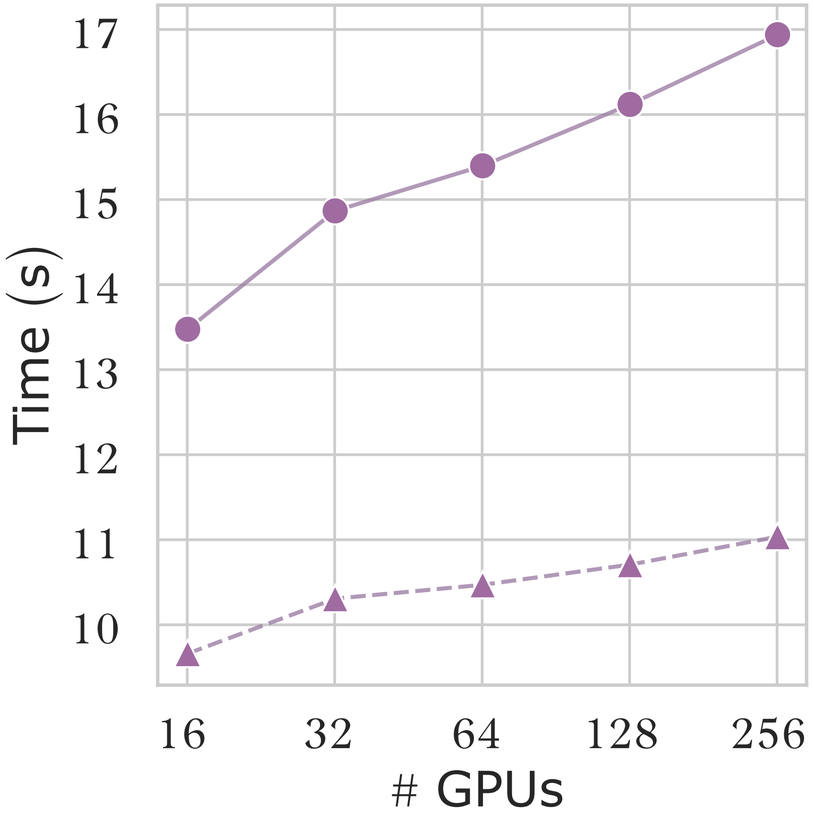
- Tests performed on *Leonardo* @CINECA
  - Each node has 4x NVIDIA A100 GPUs
  - MPI for communication across nodes
- With **64 nodes** (256 GPUs), SIGMo processed:
  - **128M molecules** from the *ZINC dataset* (500K per GPU)
  - A fixed set of **389 queries**
- We used **static partitioning** of the dataset to split the workload across GPUs
  - Runtime variation across nodes remained low (4-8%)

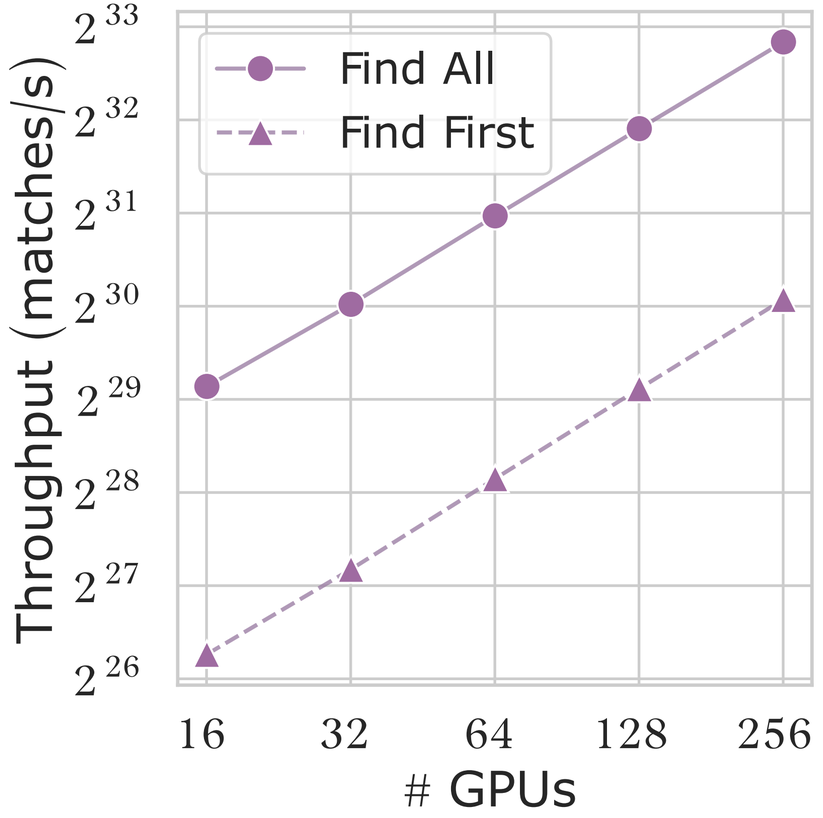
Experimental Evaluation – **Scalability Evaluation**



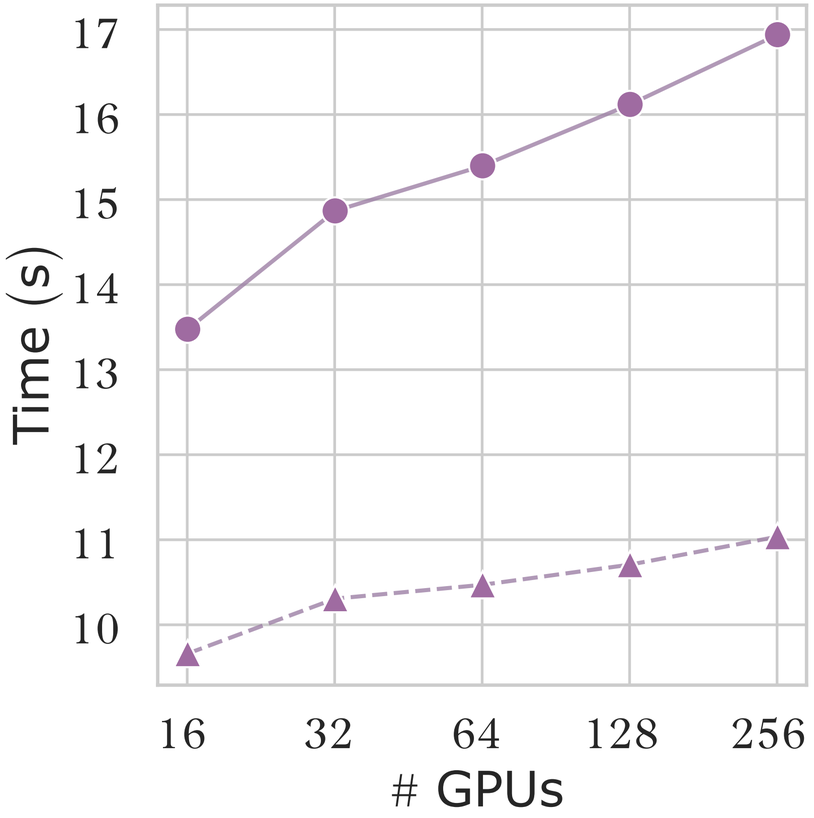


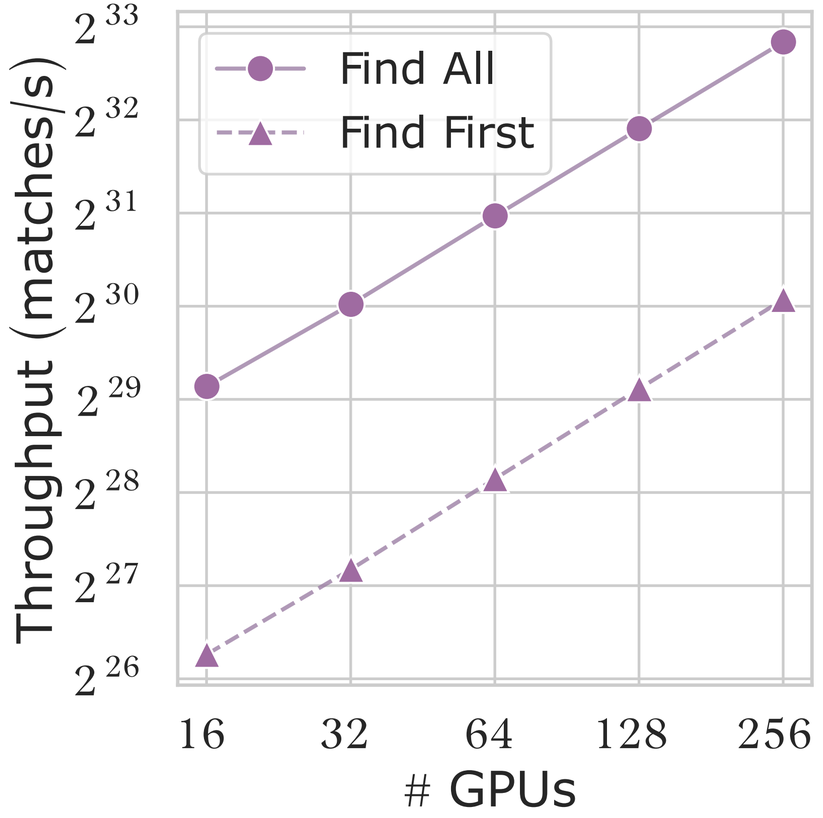
Experimental Evaluation – **Scalability Evaluation**





Experimental Evaluation – **Scalability Evaluation**





Achieved a peak throughput of **7.7B** matches/s

Conclusion

Reach us on GitHub

Conclusion

- SIGMo is a **domain-aware**, **GPU-batched** framework for molecular matching

Reach us on GitHub

Conclusion

- SIGMo is a **domain-aware**, **GPU-batched** framework for molecular matching
- Scales linearly up to 256 GPUs, reaching up to **7.7 billion matches/s** on real-world datasets

Reach us on GitHub

Conclusion

- SIGMo is a **domain-aware**, **GPU-batched** framework for molecular matching
- Scales linearly up to 256 GPUs, reaching up to **7.7 billion matches/s** on real-world datasets
- Awarded three ACM reproducibility badges

Reach us on GitHub

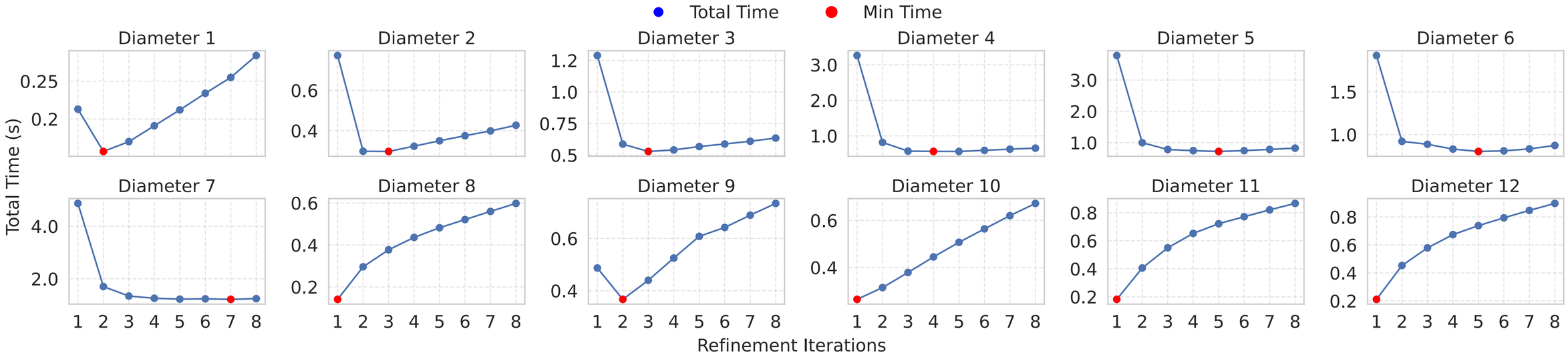
Conclusion

- SIGMo is a **domain-aware**, **GPU-batched** framework for molecular matching
- Scales linearly up to 256 GPUs, reaching up to **7.7 billion matches/s** on real-world datasets
- Awarded three ACM reproducibility badges
- Contacts:
  - Antonio De Caro antdecaro@unisa.it
  - Gennaro Cordasco gcordasco@unisa.it
  - Federico Ficarelli f.ficarelli@cineca.it
  - Biagio Cosenza bcosenza@unisa.it

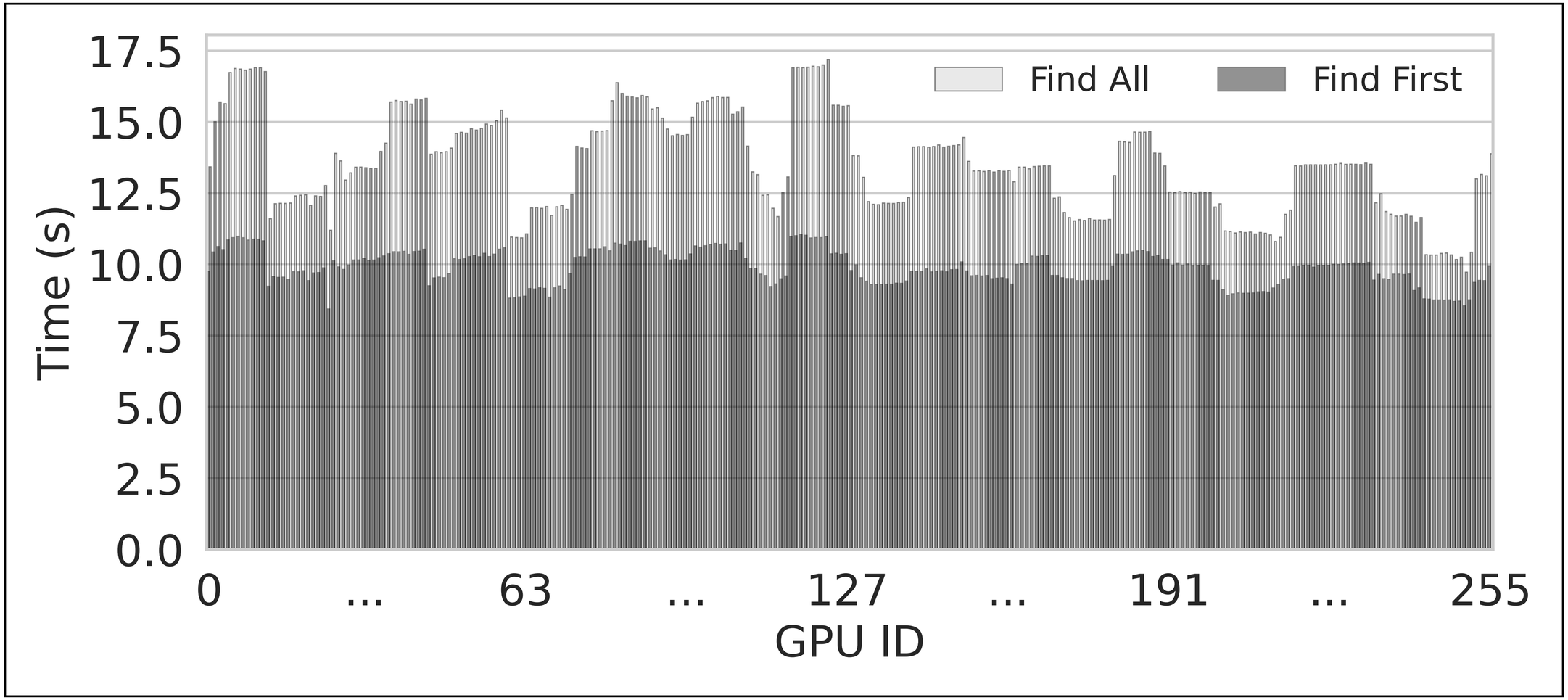
Reach us on GitHub

Backup Slides

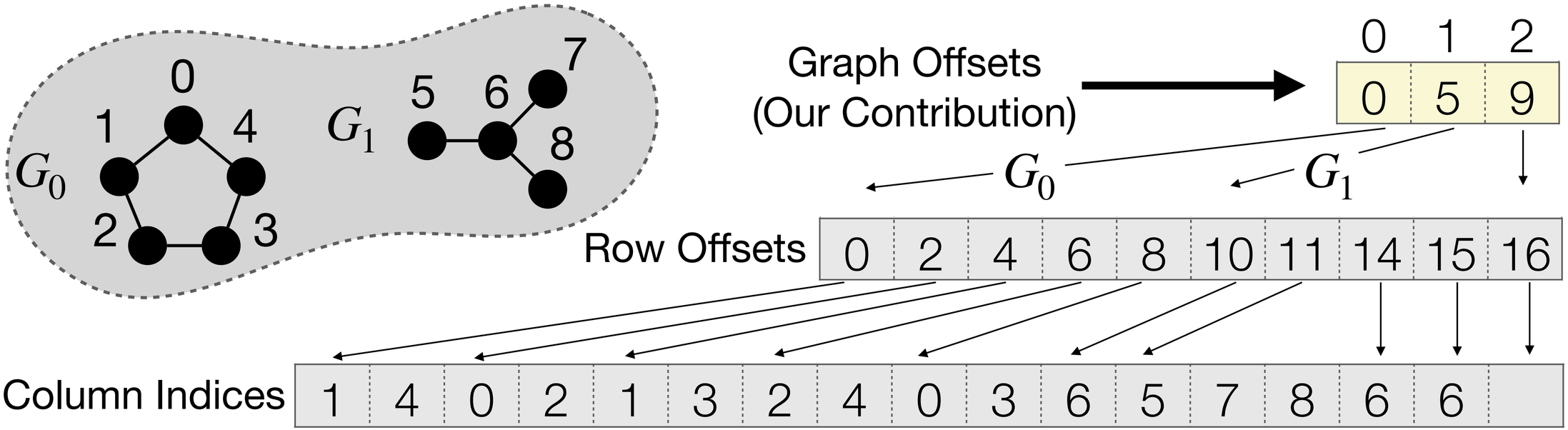
Grouping Queries by Diameter



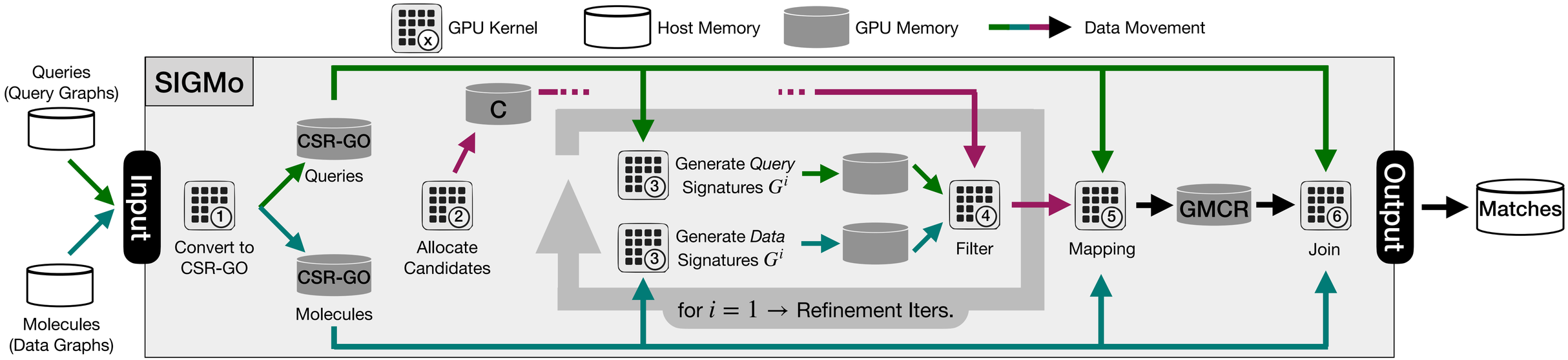
Grouping Queries by Diameter



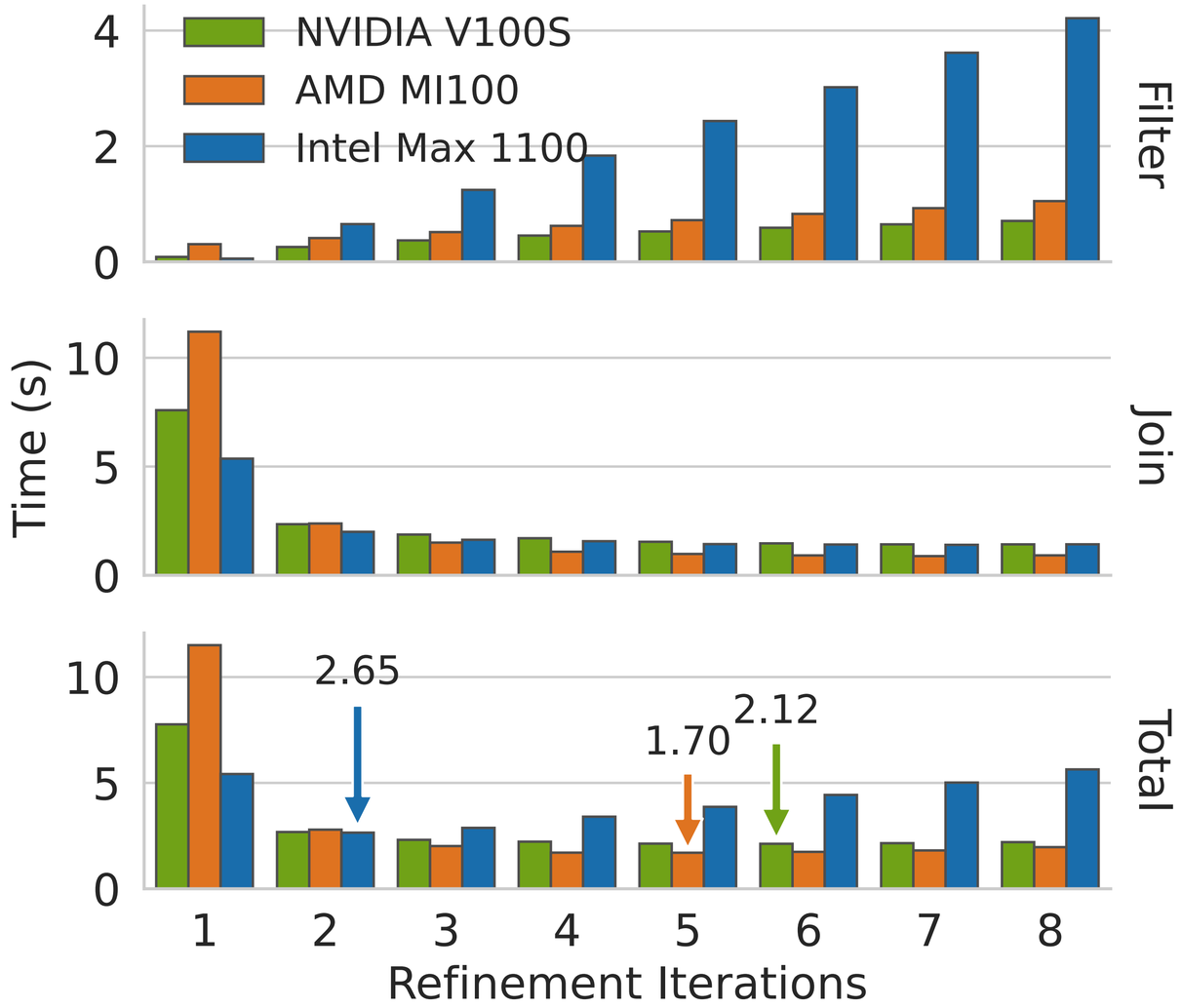
CSR-GO: Expressing Disconnected Components of a Graph



SIGMo's Pipeline



Performance Portability Assessment



Scalability on Single GPU