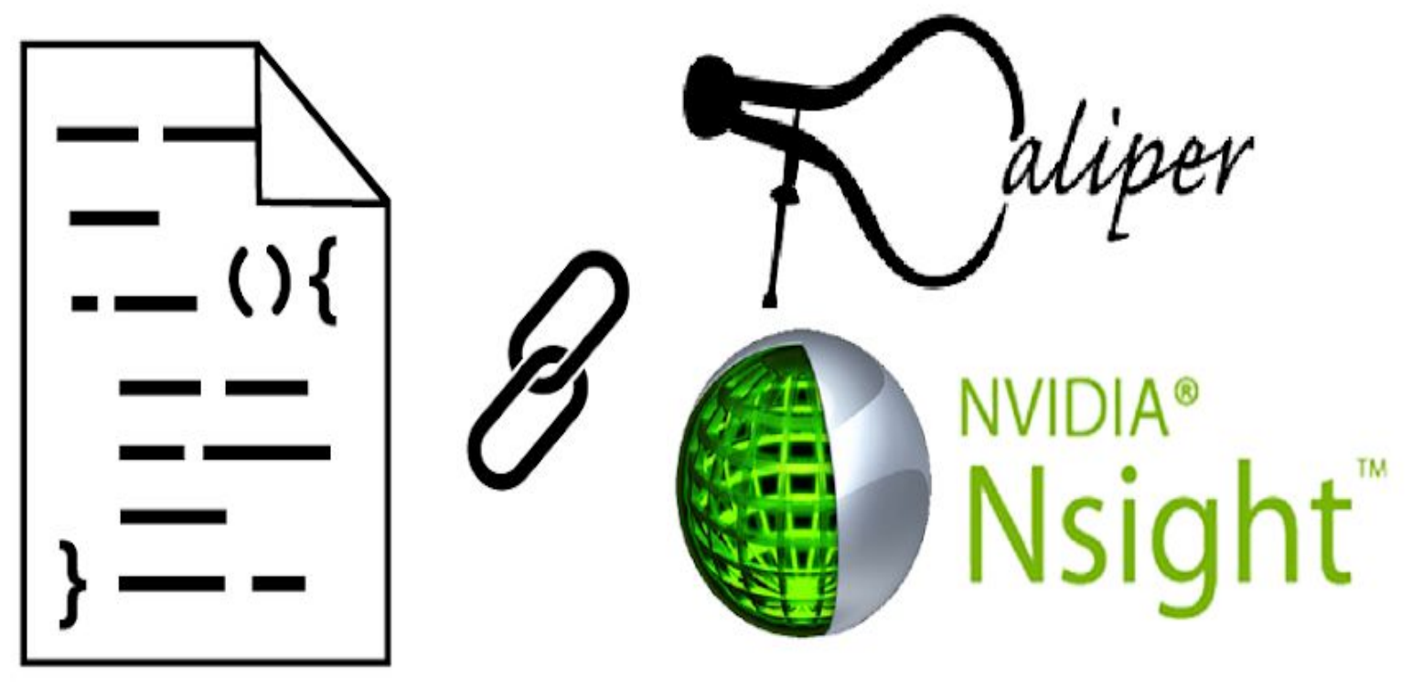
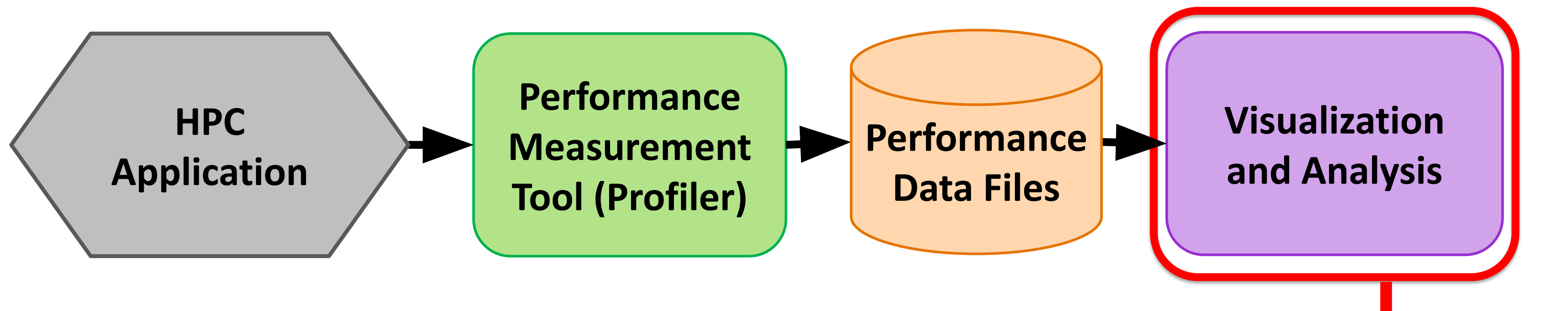
Identifying Performance Bottlenecks in Scientific Applications with Call Path Querying

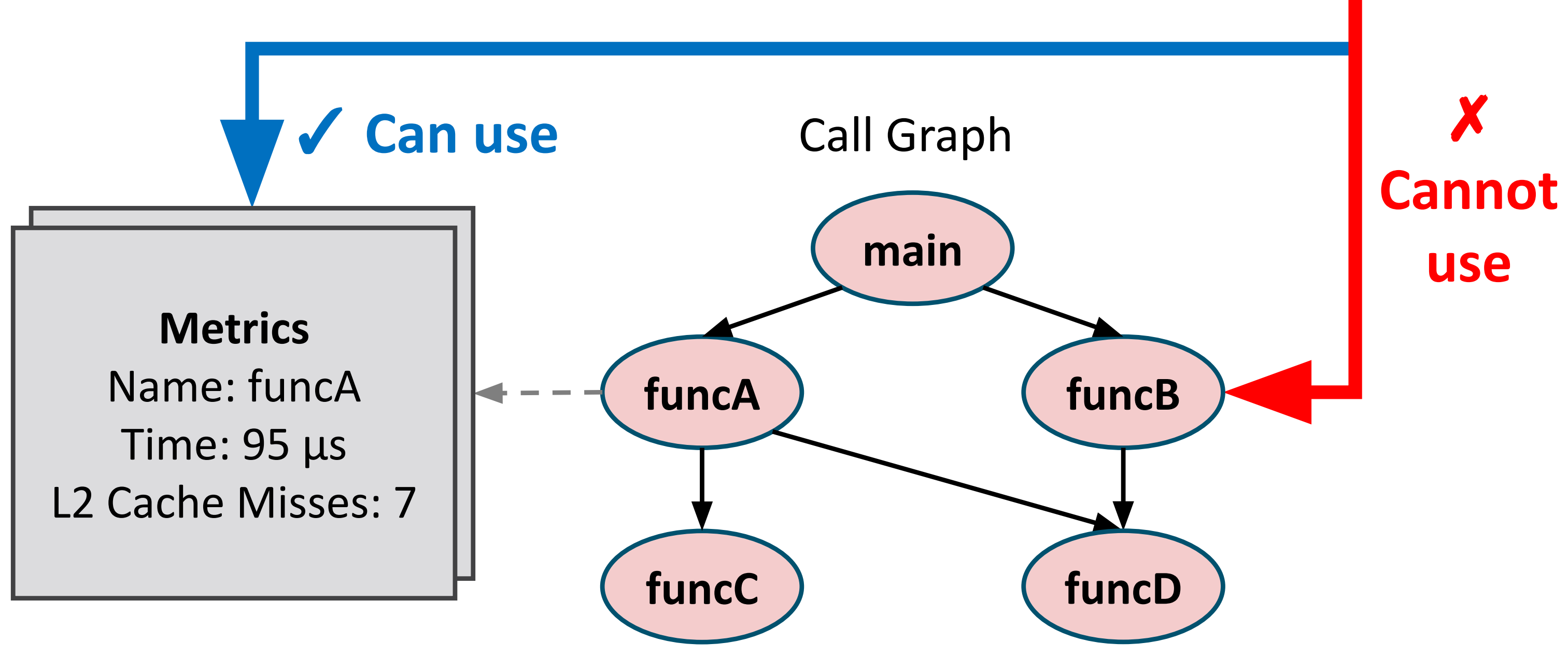
Ian Lumsden, Jakob Luettgau, Vanessa Lama, Connor Scully-Allison,
Stephanie Brink, Katherine E. Isaacs, Olga Pearce, Michela Taufer

The Challenges of Analyzing Ensembles of Performance Data



Collecting performance data

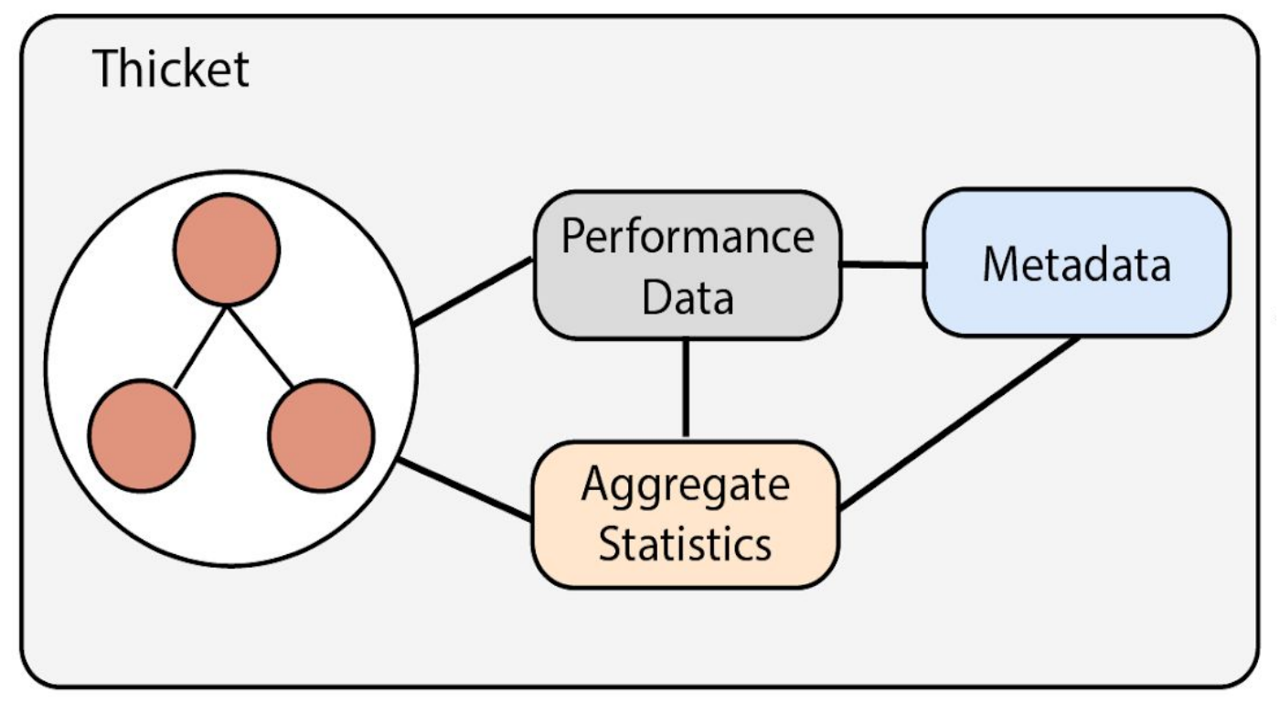

HPC Application
Performance Measurement Tool (Profiler)
Performance Data Files
Visualization and Analysis



Metrics
Name: funcA
Time: 95 µs
L2 Cache Misses: 7

- HPC applications get larger and more complex
- Performance analysis plays a critical role in understanding bottlenecks on HPC systems
- Challenges:
  - Performance analysis is limited by the design of profilers and analysis tools
  - Scientists need a more effective way to manage and analyze data from multi-run performance experiments

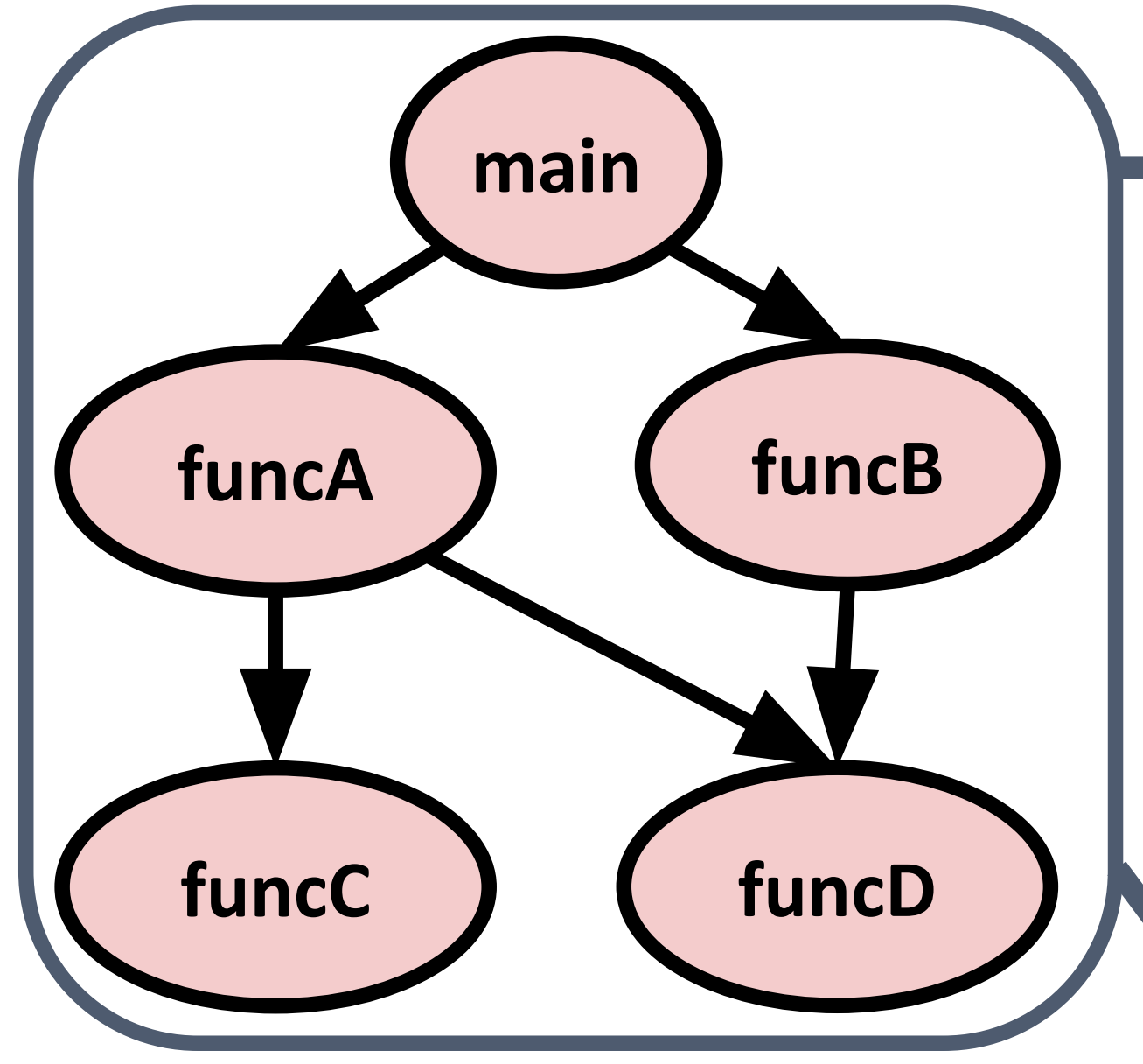
Enabling Analysis of Performance Data Ensembles



Unifying performance data

- Thicket builds on Hatchet to enable data analysis for multi-run performance experiments (ensembles)
- Thicket enables scientists to:
  - Merge multi-run data into a single performance view
  - Analyze data easily with built-in functionality and other Python tools

Call Graph



Performance data

Node	Profile ID	Time	L2 Cache Misses
main	1	50 µs	3
	2	47 µs	2
funcA	1	95 µs	7
	2	102 µs	9
...			

Metadata

Profile ID	Problem Size	Compiler
1	1200	clang
2	1200	gcc
...		

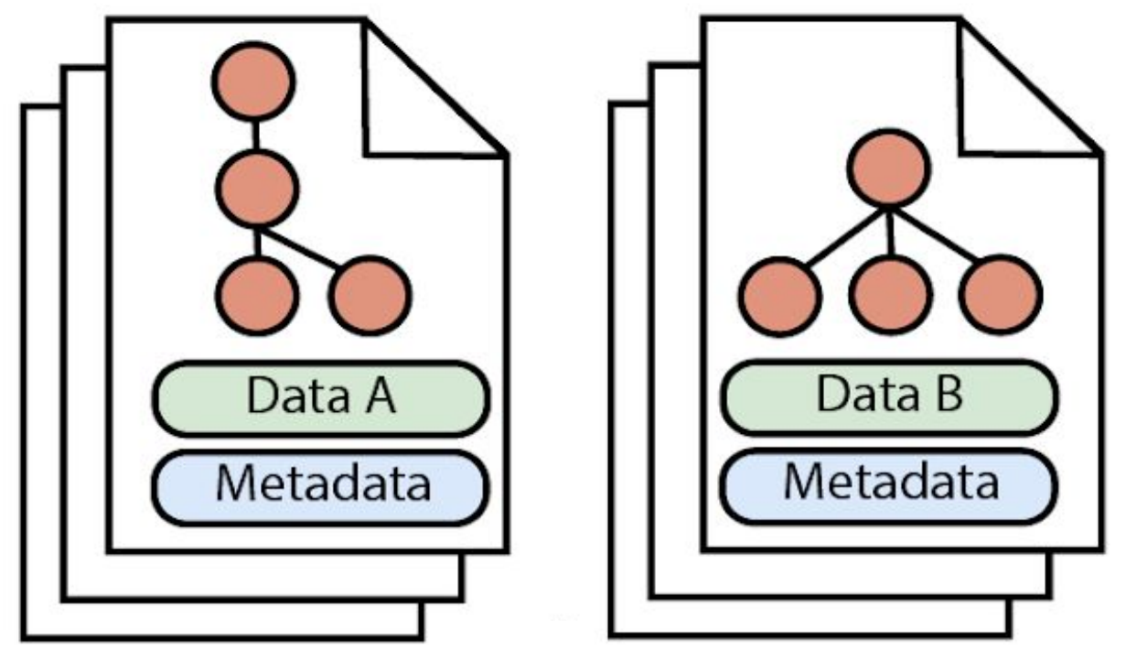
Node	Time (avg)
main	48.5 µs
funcA	98.5 µs
...	

Aggregate Statistics

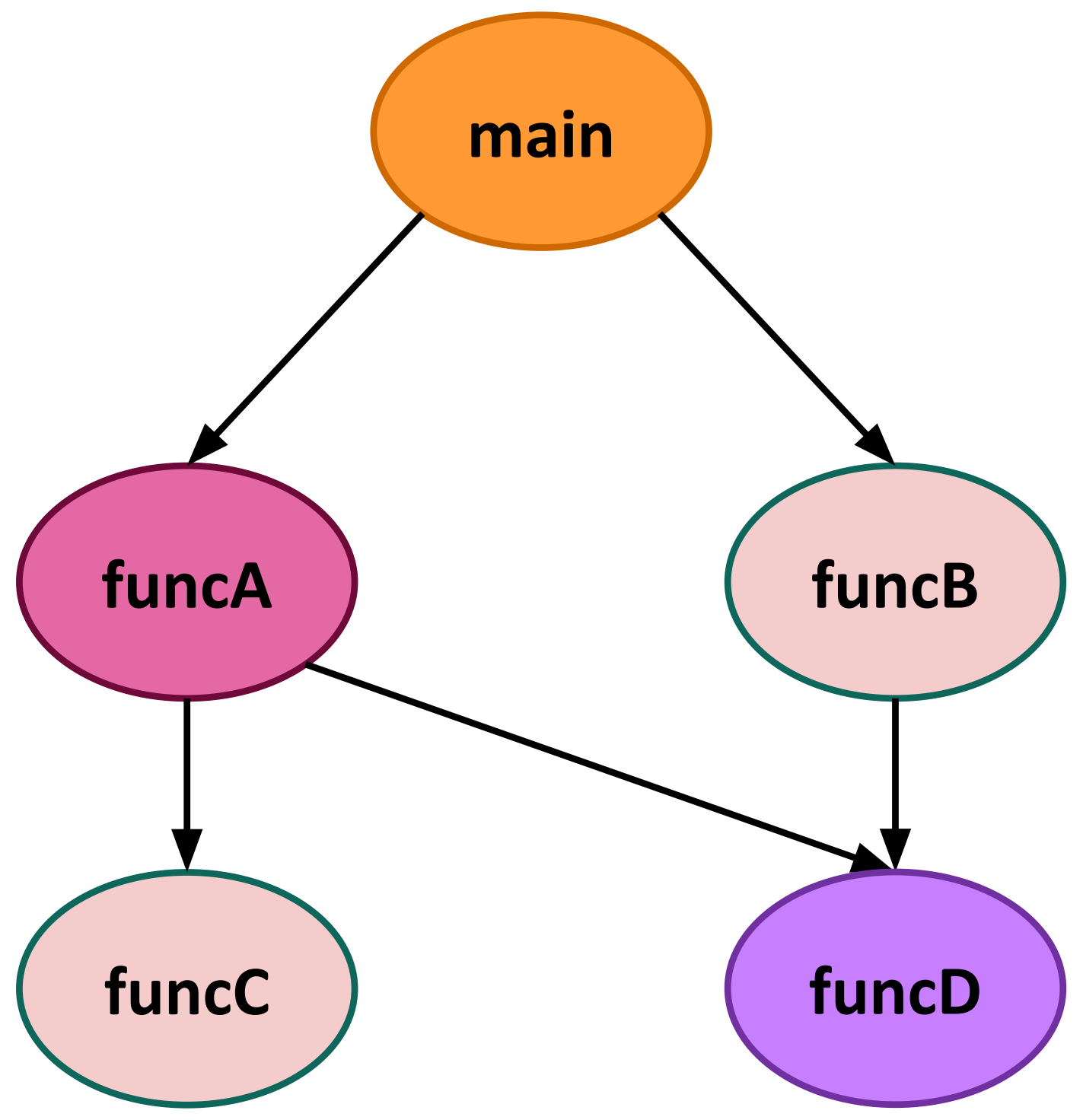
- Thicket components are:
  - Call graph
  - Performance data
  - Metadata
  - Aggregate statistics

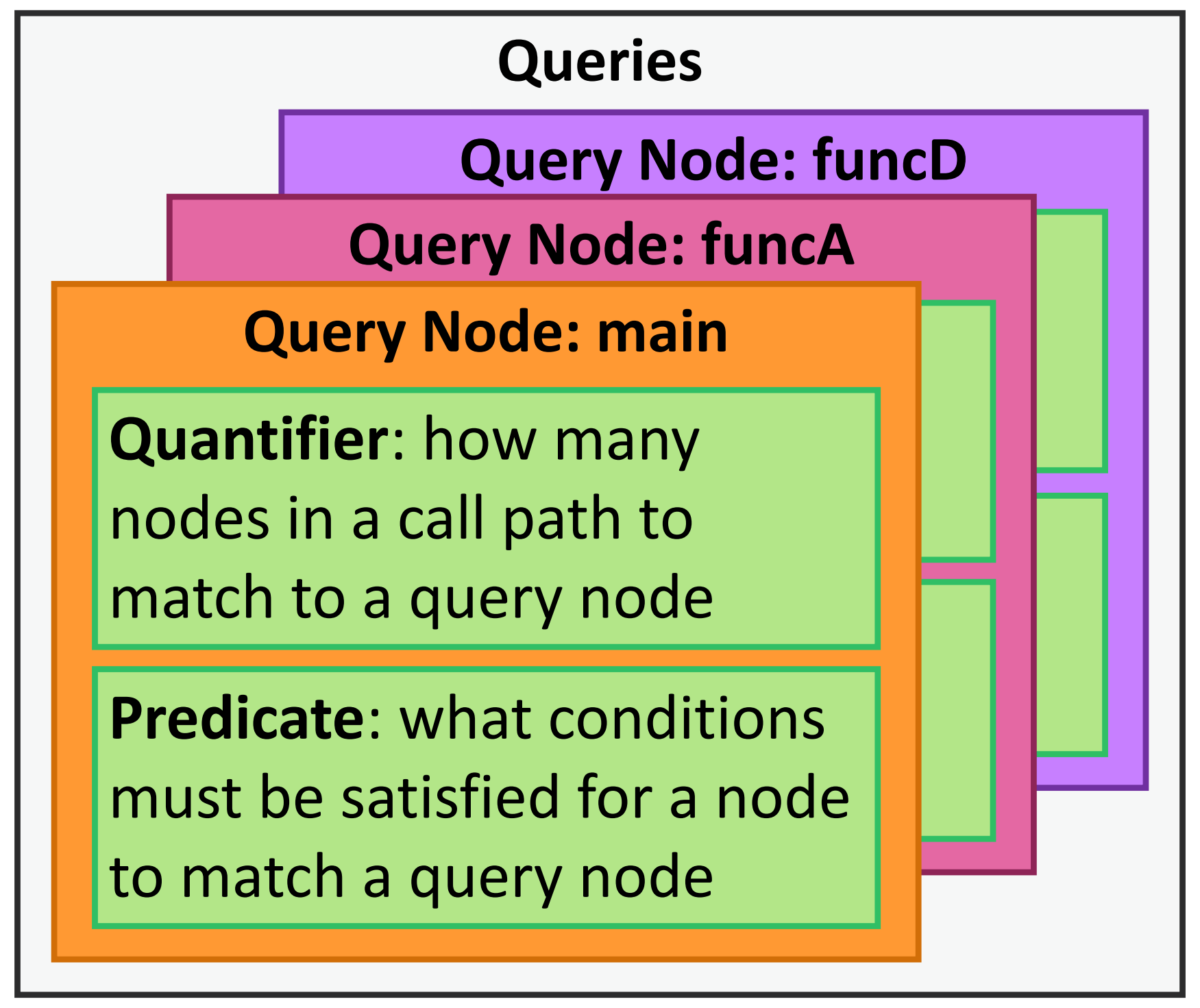
THICKET

Leveraging the Power of Call Path Querying for Performance Analysis



Augmenting metrics with call paths





Queries

Query Node: funcD

Query Node: funcA

Query Node: main

Quantifier: how many nodes in a call path to match to a query node

Predicate: what conditions must be satisfied for a node to match a query node

- We design and implement a new *Call Path Query Language* in Hatchet and Thicket
  - The language extracts paths in the call graph using *queries* (i.e., descriptions of the properties of one or more paths in the graph)
- We define two dialects for the Query Language to simplify its use under certain circumstances
  - Object-based
  - String-based
- Each dialect comes with its own strengths and weaknesses

Query Example: Find all subgraphs rooted at a MPI node with more than 5 L2 cache misses

Base Syntax

```
query = QueryMatcher().match(
     “.”,
     lambda row: re.match(
         “MPI_.*”,
         row[“name”] )
     is not None
     and row[“PAPI_L2_TCM”] > 5
  ).rel(“*”)
```

- + Support any query
- - Require Python libs knowledge
- - Work with Python only

Object-based Dialect

```
query = [
    (“.”, {
     “name”: “MPI_.*”,
     “PAPI_L2_TCM”: “> 5”
    }),
    “*”
]
```

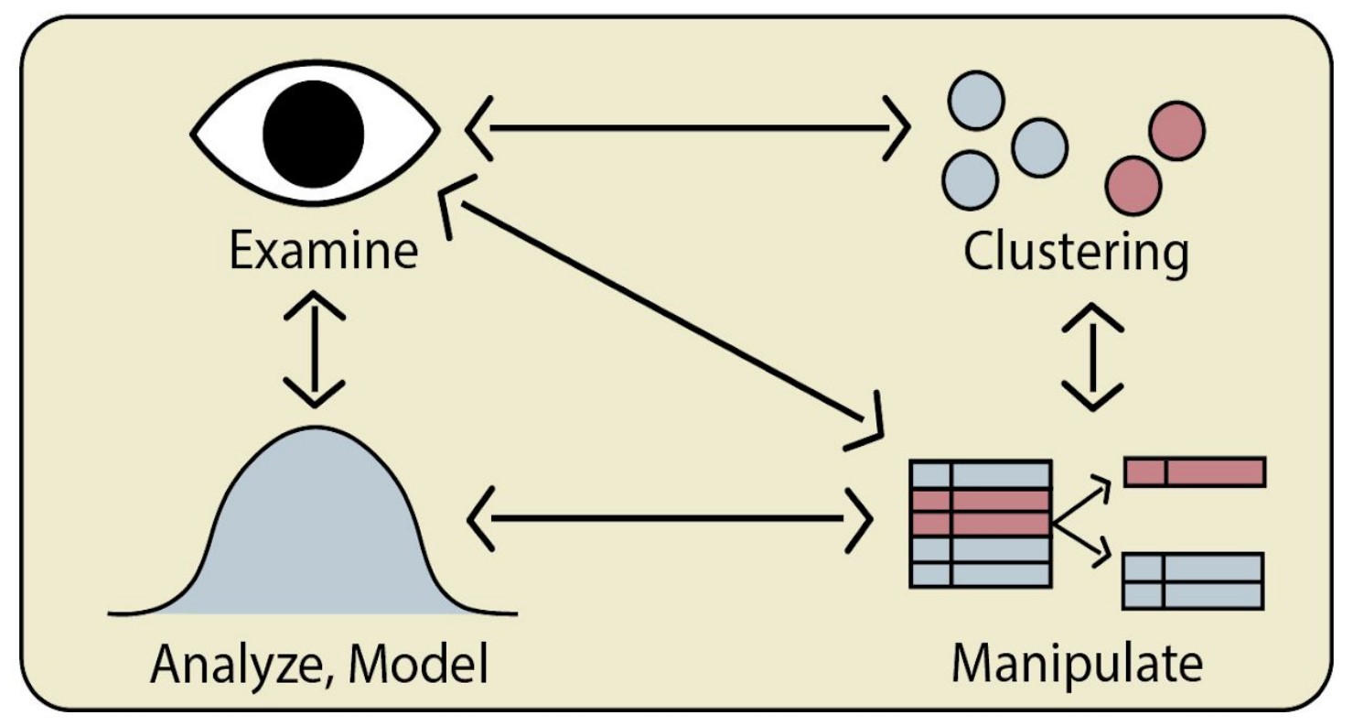
- + Use built-in Python objects
- - Support limited queries
- - Work with Python only

String-based Dialect

```
query = """
MATCH (".", p)->("*")
WHERE p."name" =~ "MPI_.*"
AND p."PAPI_L2_TCM" > 5
"""
```

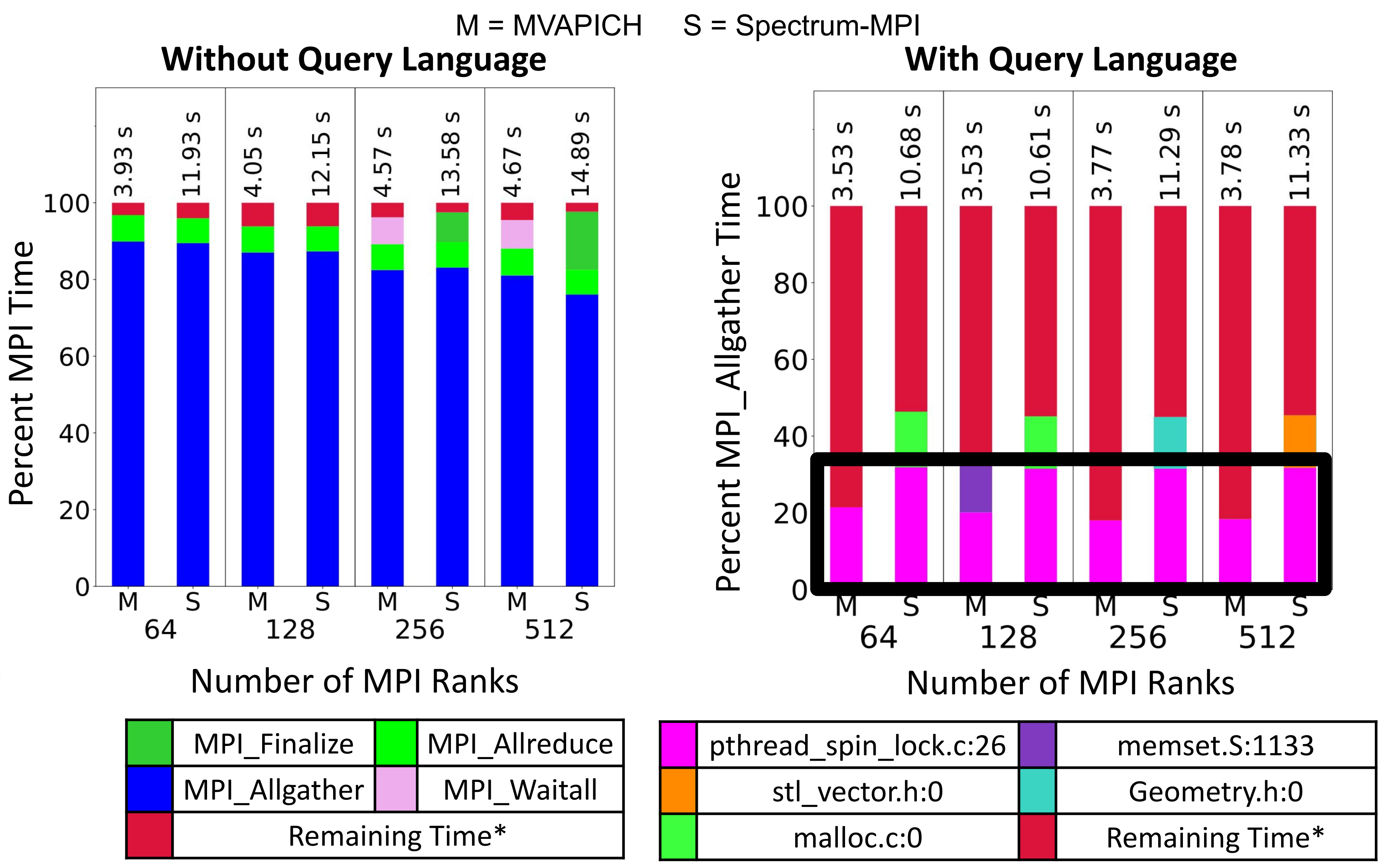
- + Work with any language
- - Support limited queries

Studying MPI Performance with Call Path Querying



Extracting knowledge from data

- We run the AMG2013 benchmark with different MPI libraries and at varying scales
  - MVAPICH and Spectrum-MPI
- We use LLNL’s Lassen (POWER9 CPU)
- We profile the runs with HPCToolkit
- Our solution allows us to look at the performance of the internals of a specific MPI call to discover potential causes of Spectrum-MPI’s worse performance



* “Remaining Time” categories = functions that take up less than 5% of the total time considered

Lesson Learned and Future Directions

- Our Query Language allows scientists to discover new insights into their applications’ performance
  - We discover that Spectrum-MPI spends a higher percentage of its MPI_Allgather time in pthread_spin_lock (roughly 30%) than MVAPICH (roughly 20%), which could explain Spectrum-MPI’s worse performance
- Future work: Deploy Thicket, augmented with our Query Language, to study the performance of *in situ* workflows (e.g., workflows for studying protein structure changes)

If you want to learn more about the Query Language, check out our eScience 2022 paper:

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory (LLNL) under Contract DE-AC52-07NA27344 (LLNL-POST-847739).
The UTK group acknowledges the support of NSF (#1841758, #1900888, and #2138811).
The authors thank Dr. Abhinav Bhatele (University of Maryland), Dr. Todd Gamblin (LLNL), and Sanjukta Bhowmick (University of Northern Texas) for their feedback during the preparation of this work.