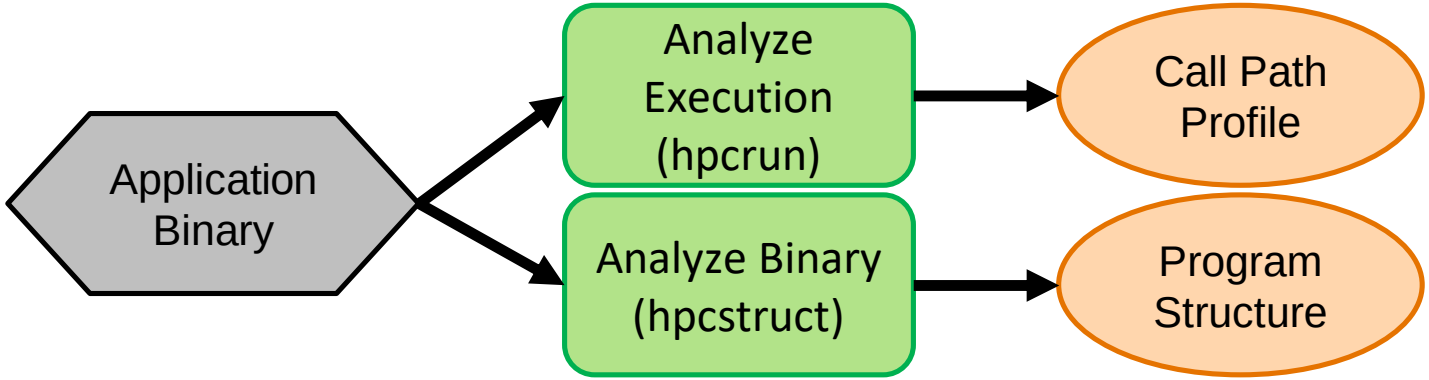
Enabling Graph-Based Profiling Analysis using Hatchet

Ian Lumsden, Stephanie Brink, Michael R. Wyatt II, Abhinav Bhatele, Olga Pearce, Todd Gamblin, Michela Taufer

Limits of HPC Profilers: HPCToolkit (I)

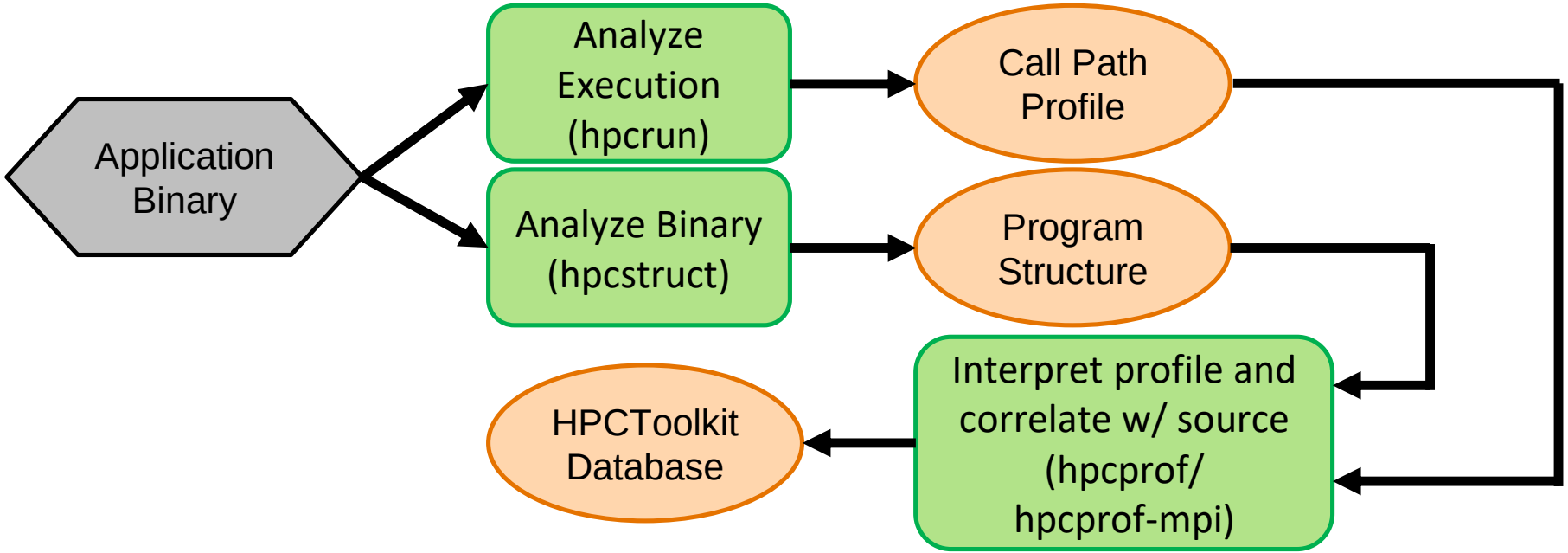
[2] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. Tallent. 2010. HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 285-701

Limits of HPC Profilers: HPCToolkit (II)



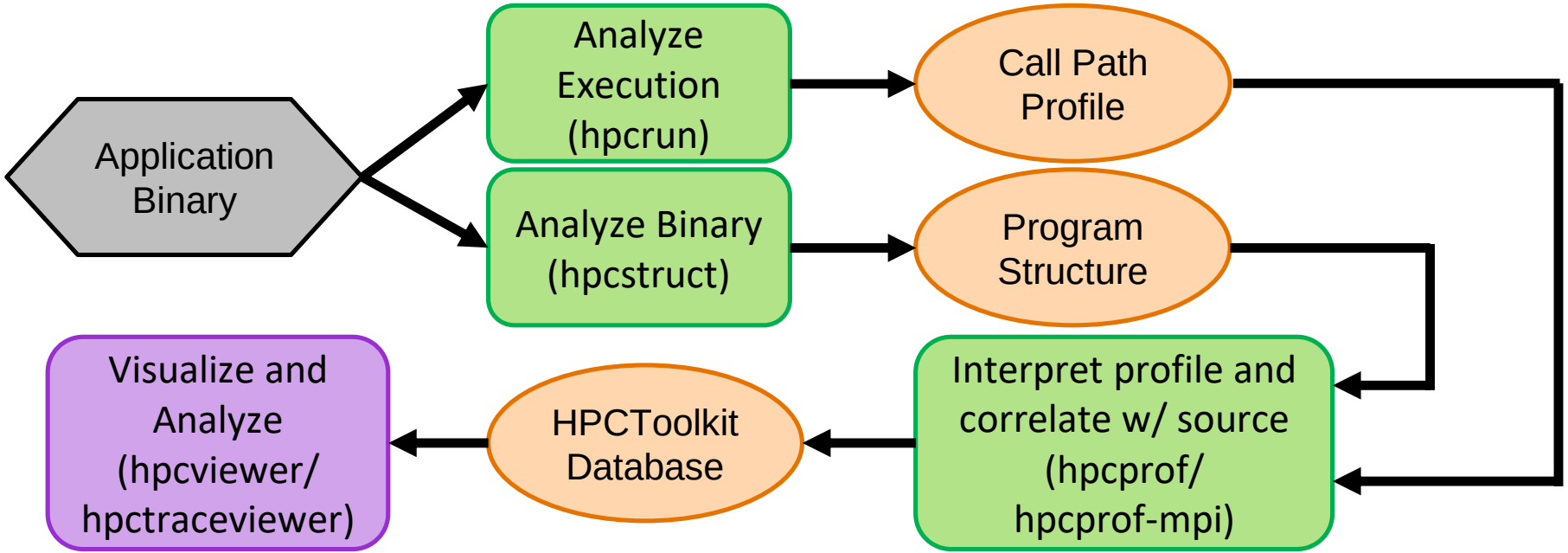
[2] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. Tallent. 2010. HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 285-701

Limits of HPC Profilers: HPCToolkit (III)



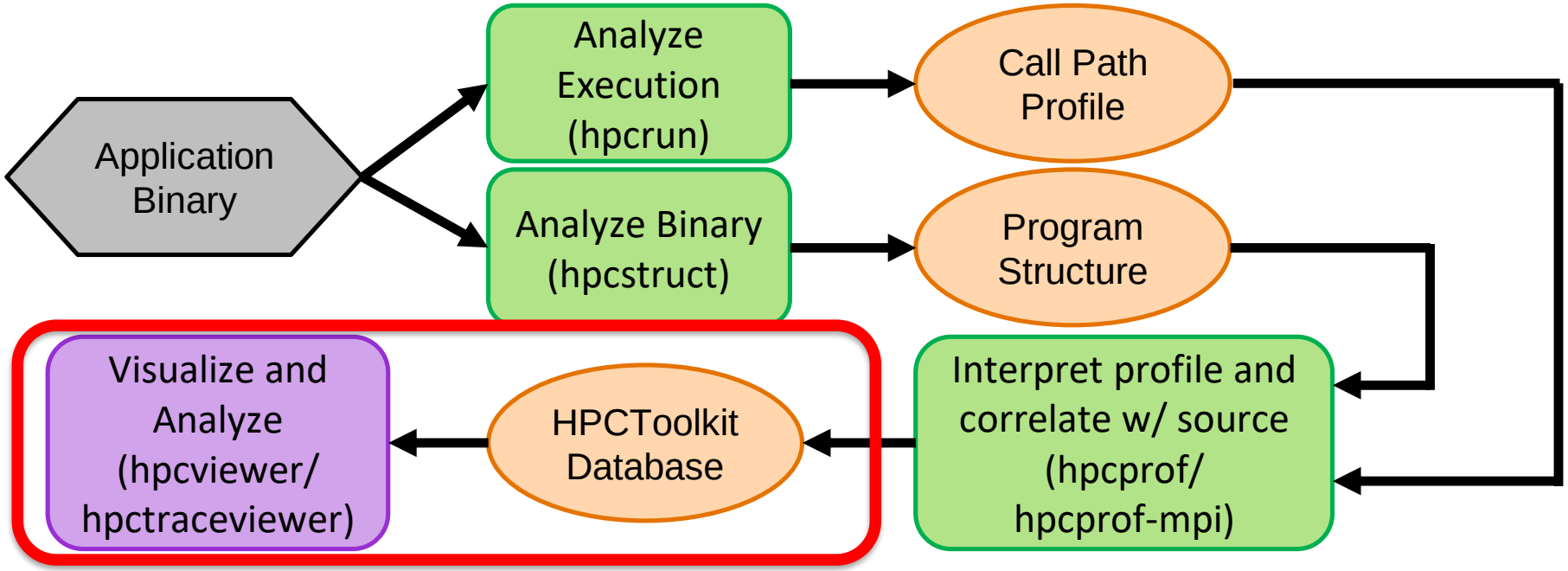
[2] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. Tallent. 2010. HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 285-701

Limits of HPC Profilers: HPCToolkit (IV)



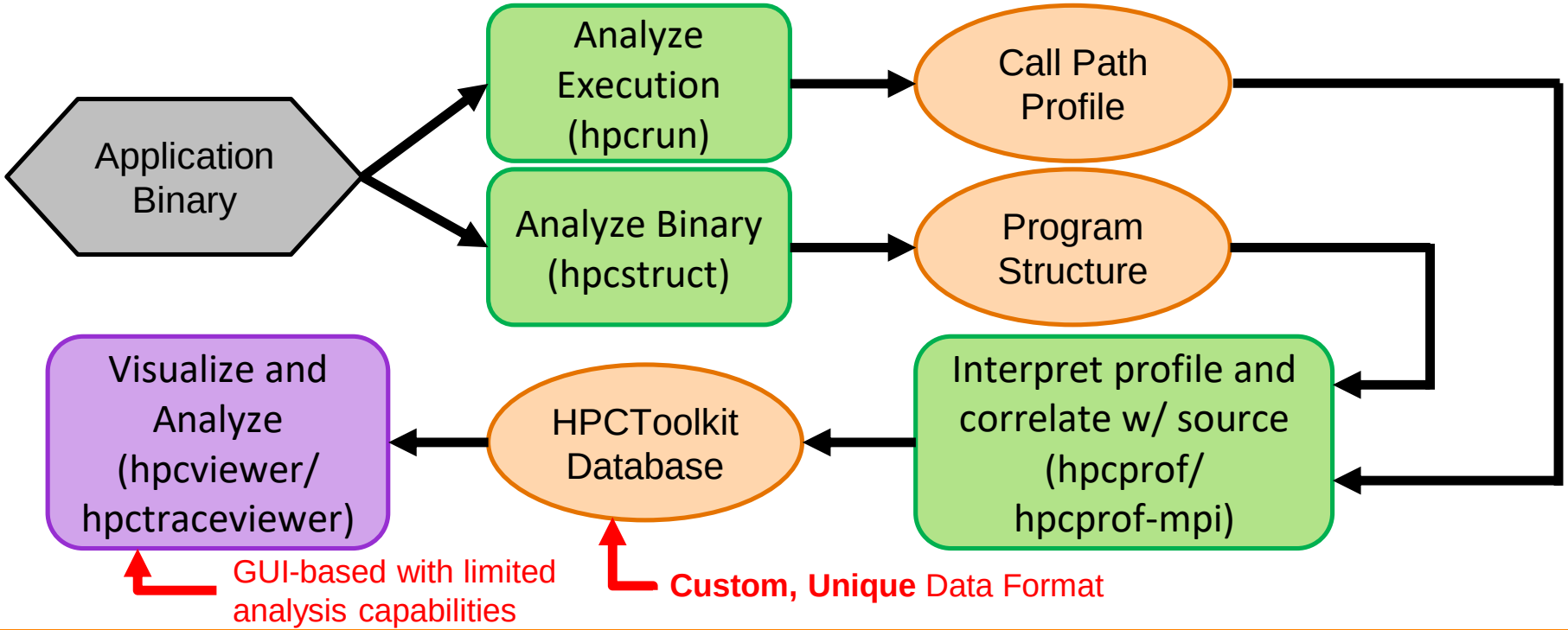
[2] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. Tallent. 2010. HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 285-701

Limits of HPC Profilers: HPCToolkit (V)



[2] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. Tallent. 2010. HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 285-701

Limits of HPC Profilers: HPCToolkit (VI)



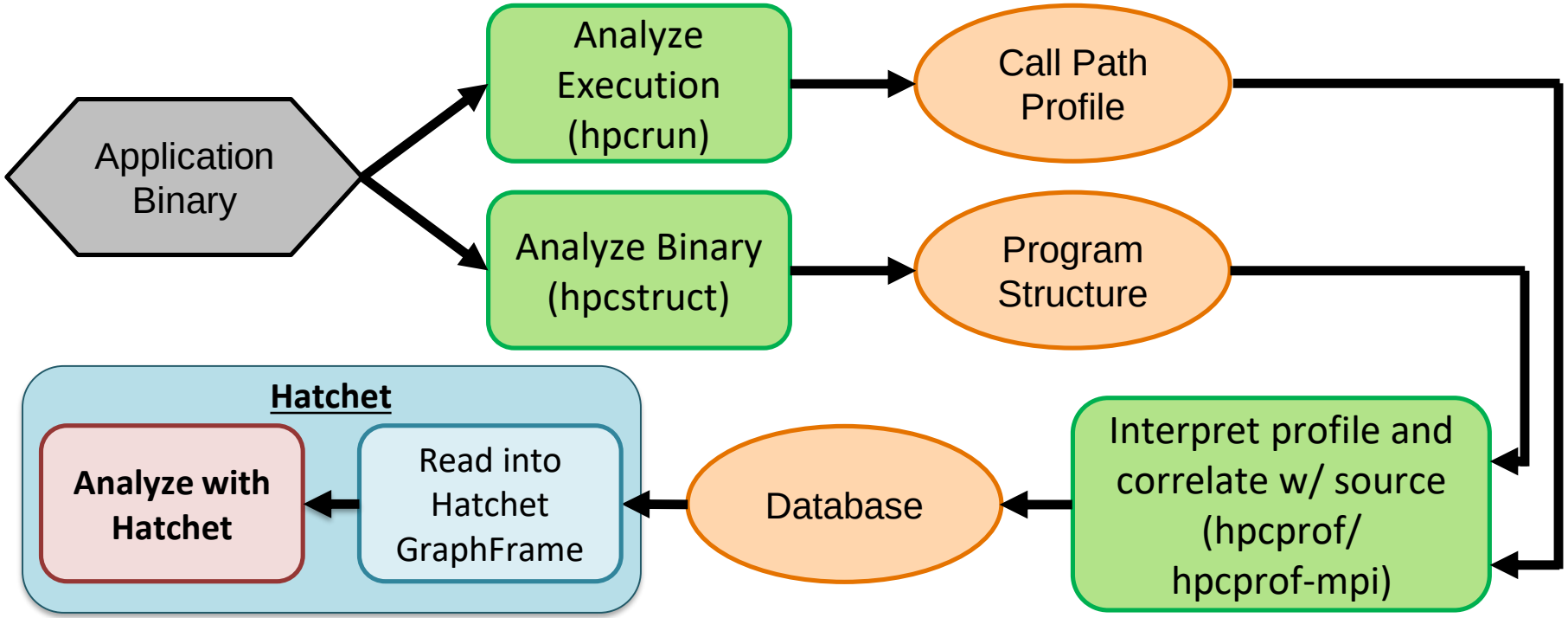
[2] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. Tallent. 2010. HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 285-701

Hatchet

- Python-based library
- A new, **general** data analysis tool that can read HPC profiling data from **different profilers**
  - Store the raw performance data into a *pandas* DataFrame
  - Represent the relational *caller-callee* data with directed acyclic graph (DAG)

[1] A. Bhatele, S. Brink, and T. Gamblin. 2019. Hatchet: Pruning the Overgrowth in Parallel Profiles. In *Proc. of the International Conference of High Performance Computing, Networking, Storage, and Analysis (SC19).*

Hatchet



Hatchet

- Python-based library
- A new, **general** data analysis tool that can read HPC profiling data from **different profilers**
  - Store the raw performance data into a *pandas* DataFrame
  - Represent the relational *caller-callee* data with directed acyclic graph (DAG)

- Hatchet restricts users to table-based analysis of the raw performance data
- Hatchet does NOT support analysis of the relational data collected by HPC profilers

[1] A. Bhatele, S. Brink, and T. Gamblin. 2019. Hatchet: Pruning the Overgrowth in Parallel Profiles. In *Proc. of the International Conference of High Performance Computing, Networking, Storage, and Analysis (SC19).*

Research Goals

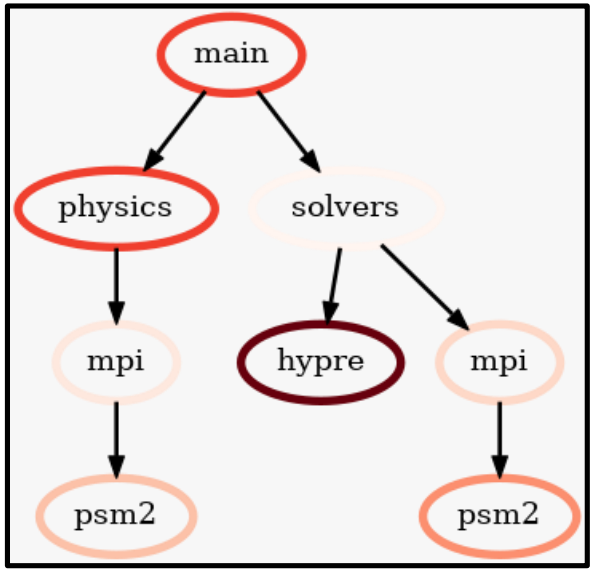
- Design new graph-based filtering query language to enable the use of relational data collected by profilers in analysis
- Implement this query language and integrate it into Hatchet analysis
- Use the augmented Hatchet to analyze the performance of different MPI calls in HPC benchmark applications

Hatchet Query Language

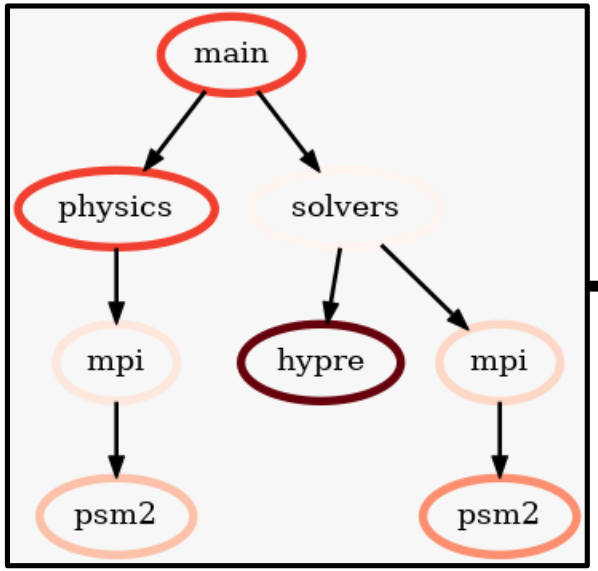
Our graph-based filtering query language consists of:

- User Input: A Query Path represented as a list of abstract graph nodes
- Algorithm:
  1. Read and parse the user's query path
  2. Match real nodes in the graph being filtered to the abstract nodes in the query path
  3. Collect all graph paths that match the full query path
  4. Create a new graph containing only the nodes in the matched paths

Hatchet Query Language: Example

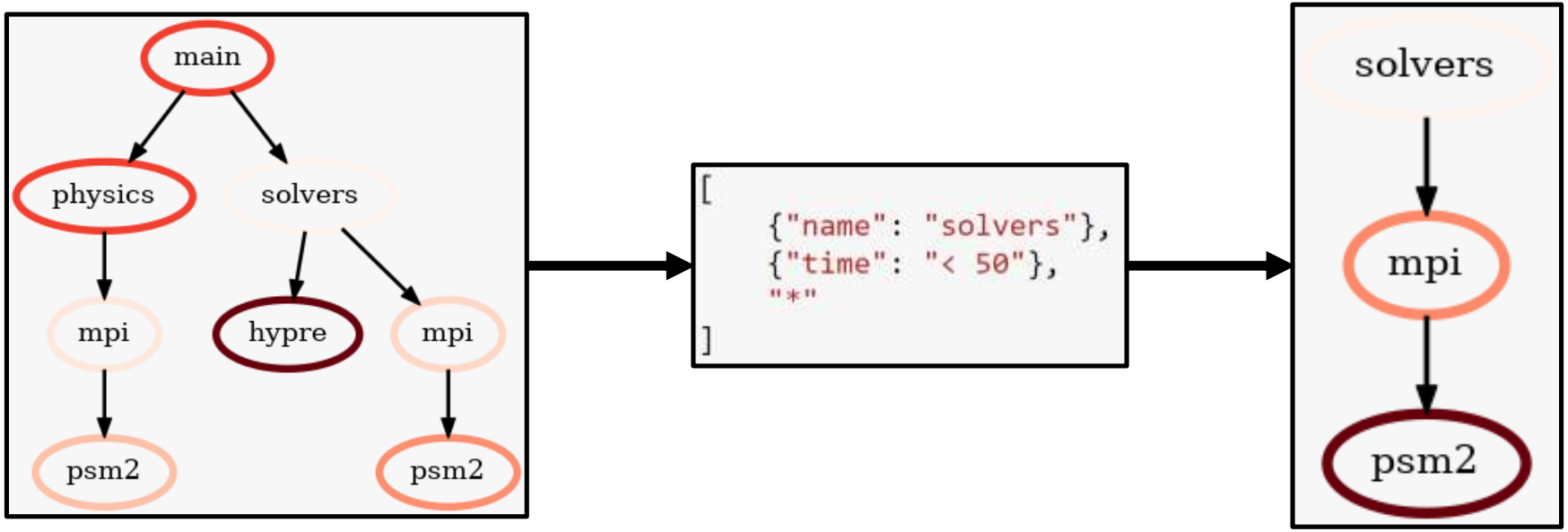


Hatchet Query Language: Example

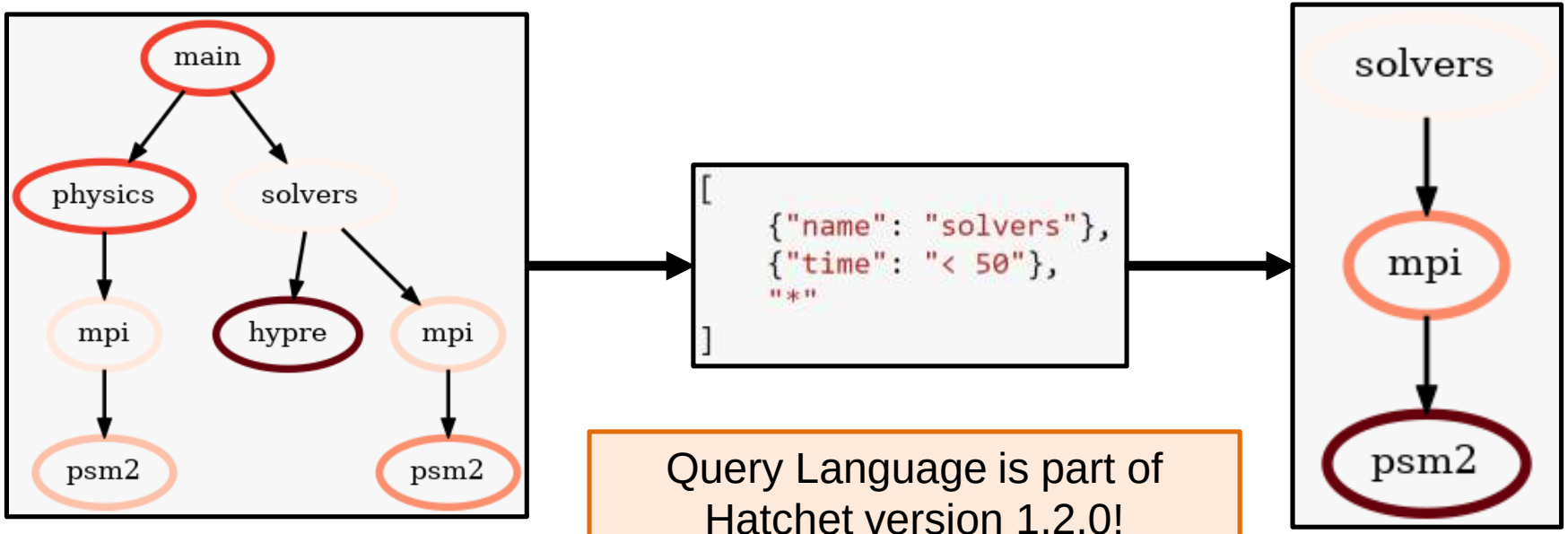



```
[
    {"name": "solvers"},
    {"time": "< 50"},
    "*"
]
```

Hatchet Query Language: Example



Hatchet Query Language: Example



Query Language is part of Hatchet version 1.2.0!

Case Study: Collect Data (I)

- Run each of the following benchmarks with MVAPICH (M) and Spectrum-MPI (S) using 64, 128, 256, and 512 MPI ranks, and profile the runs with HPCToolkit
  - AMG2013
  - Kripke
  - Lammps
- Load the generated profiles into Hatchet

Case Study: Collect Data (II)

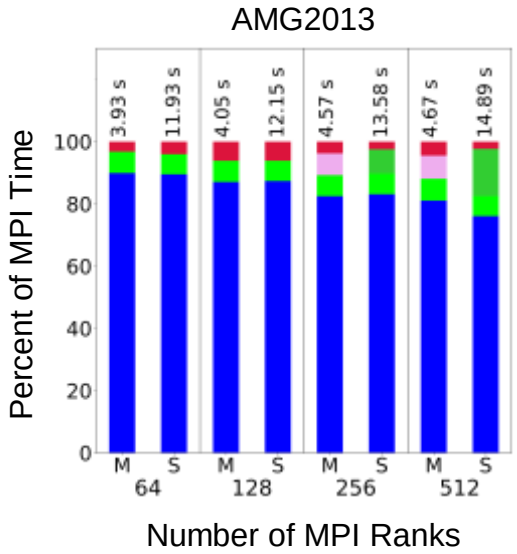
- Perform the benchmarking on LLNL's Lassen supercomputer
  - LLNL CZ (Public Collaboration) CORAL Supercomputer
  - 795 AC922 Nodes
    - 2 IBM Power9 CPUs per node (20 cores per node)
    - 256 GB Memory per node
  - Infiniband EDR interconnect

Case Study: Extract MPI Layer

- Filter the call graphs to get subgraphs rooted at standard MPI function calls
- Query Path used to extract MPI Layer:

```
[{“name”: “P?MPI_.*”}, “*”]
```

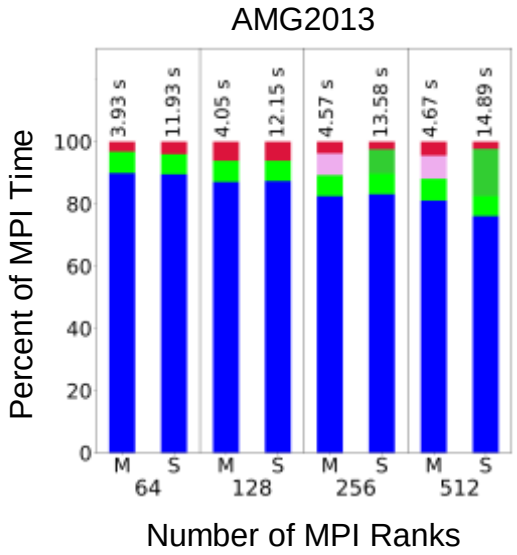
Case Study: Calculate Percent MPI Time for each MPI Function

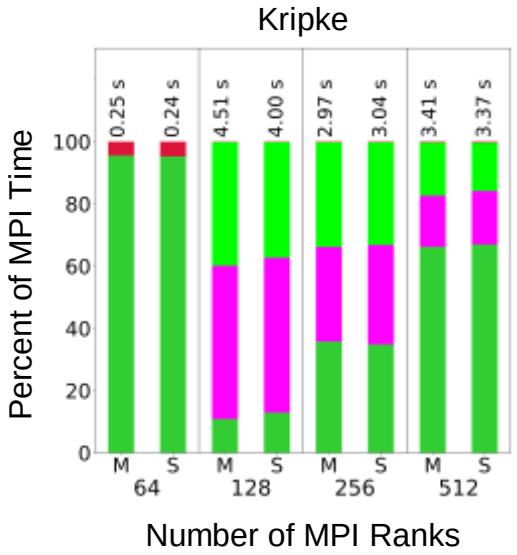


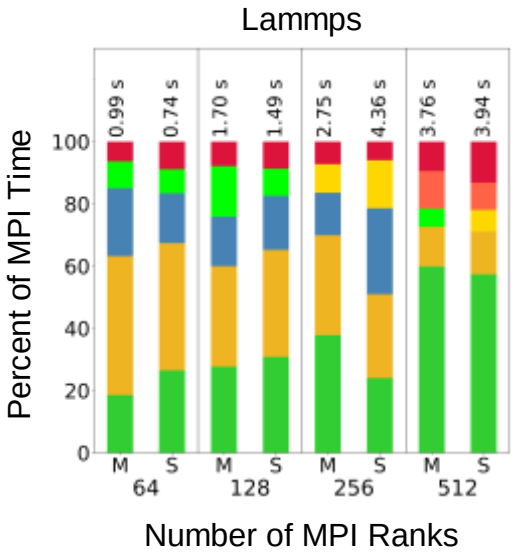
Only MPI functions that contribute at least 5% of total MPI time are shown

MPI Function Calls			
	MPI_Finalize		MPI_Send
	MPI_Allreduce		MPI_Wait
	MPI_Allgather		MPI_Waitany
	MPI_Waitall		MPI_Alltoallv
	MPI_Testany		Remaining MPI Time

Case Study: Calculate Percent MPI Time for each MPI Function



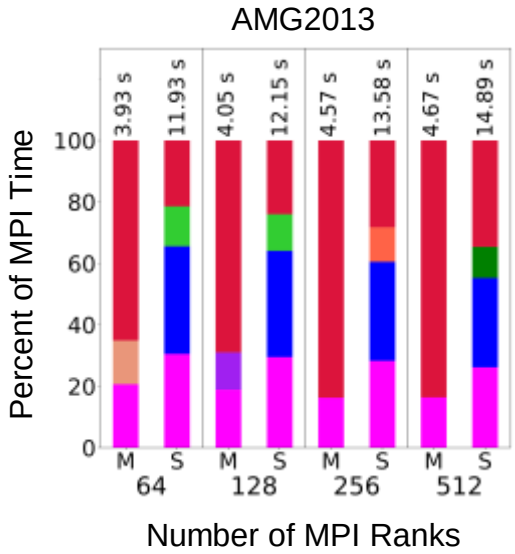




MPI Function Calls			
	MPI_Finalize		MPI_Send
	MPI_Allreduce		MPI_Wait
	MPI_Allgather		MPI_Waitany
	MPI_Waitall		MPI_Alltoallv
	MPI_Testany		Remaining MPI Time

Only MPI functions that contribute at least 5% of total MPI time are shown

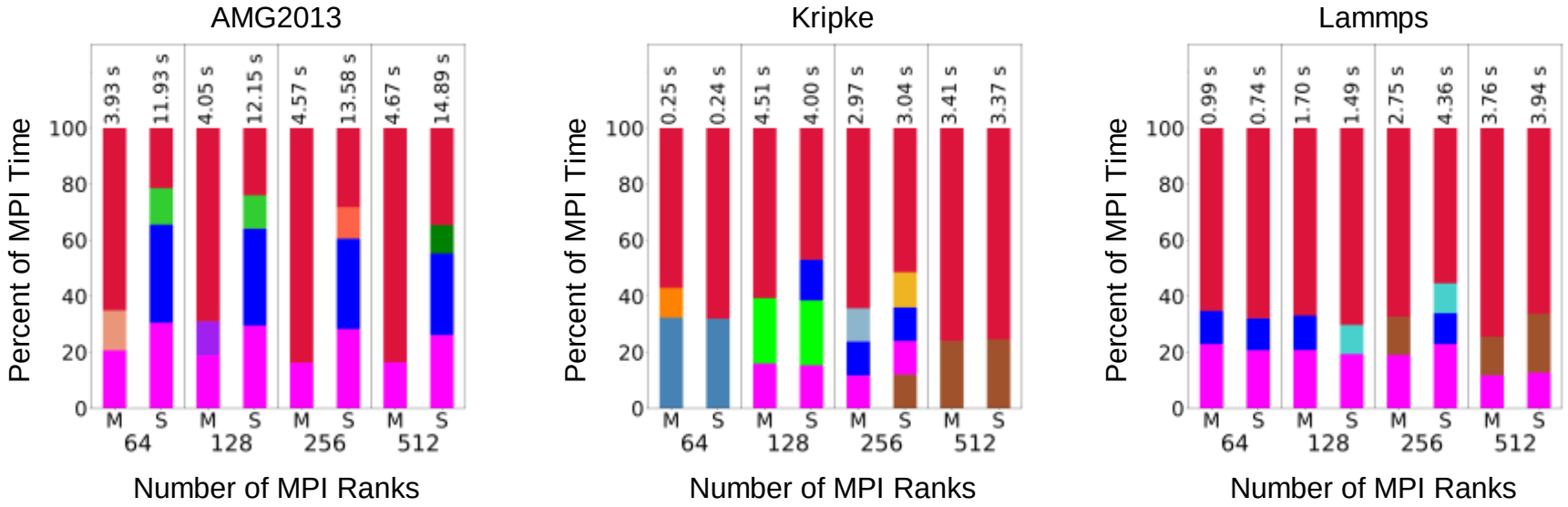
Case Study: Calculate Percent MPI Time for Child Calls of MPI Functions



Child Function Calls					
	<unknown file> [libopen-pal.so.3.1.0]:0		<unknown file> [libmlx5.so.1.0.0]:1133		<unknown file> [libmlx5.so.1.0.0]:0
	<unknown file> [libpami.so.3.1.0]:0		pthread_spin_lock.c:26		syscall-template.S:81
	syscall-template.S:82		pml_pami_send.c:0		pml_pami_init.c:0
	cancellation.c:81		memset.S:1133		Geometry.h:0
	stl_vector.h:0		malloc.c:0		Remaining MPI Time

Only children functions that contribute at least 10% of total MPI time are shown

Case Study: Calculate Percent MPI Time for Child Calls of MPI Functions

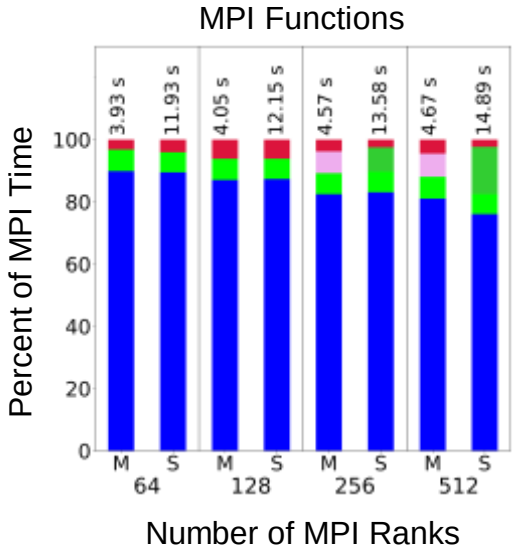


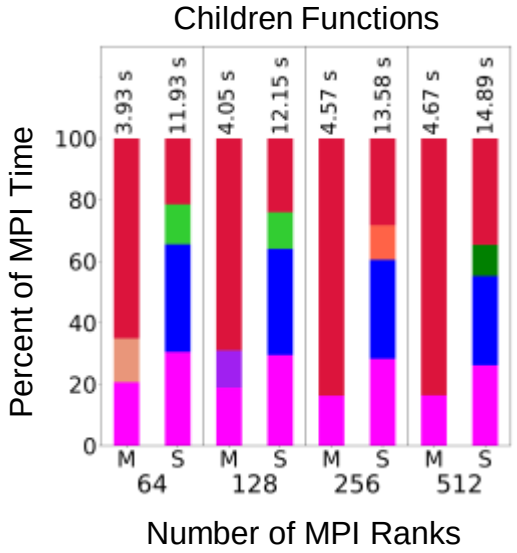
Child Function Calls					
	<unknown file> [libopen-pal.so.3.1.0]:0		<unknown file> [libmlx5.so.1.0.0]:1133		<unknown file> [libmlx5.so.1.0.0]:0
	<unknown file> [libpami.so.3.1.0]:0		pthread_spin_lock.c:26		syscall-template.S:81
	syscall-template.S:82		pml_pami_send.c:0		pml_pami_init.c:0
	cancellation.c:81		memset.S:1133		Geometry.h:0
	stl_vector.h:0		malloc.c:0		Remaining MPI Time

Only children functions that contribute at least 10% of total MPI time are shown

Case Study: Identify Slow-Down Causes

Zoom into specific benchmarks (**AMG2013**) and examine the children of specific MPI functions (**MPI_Allgather**)



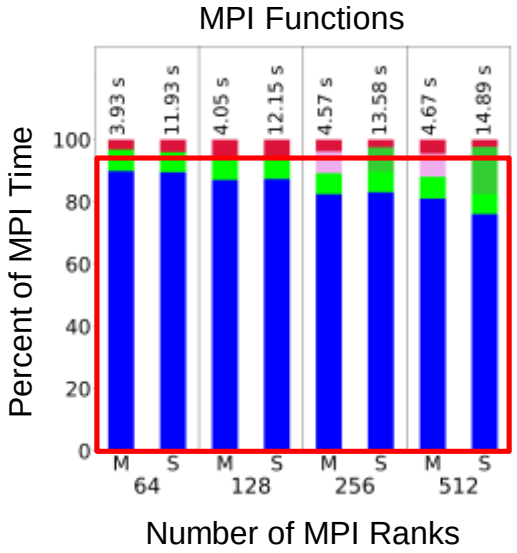


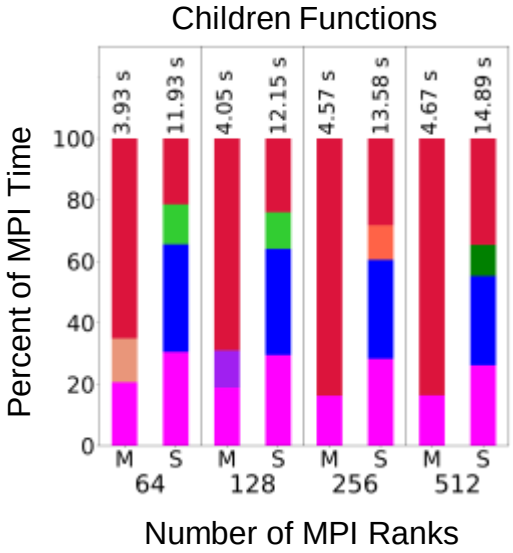
MPI Function Calls			
	MPI_Finalize		MPI_Send
	MPI_Allreduce		MPI_Wait
	MPI_Allgather		MPI_Waitany
	MPI_Waitall		MPI_Alltoallv
	MPI_Testany		Remaining MPI Time

Child Function Calls					
	<unknown file> [libopen-pal.so.3.1.0]:0		<unknown file> [libmlx5.so.1.0.0]:1133		<unknown file> [libmlx5.so.1.0.0]:0
	<unknown file> [libpami.so.3.1.0]:0		pthread_spin_lock.c:26		syscall-template.S:81
	syscall-template.S:82		pml_pami_send.c:0		pml_pami_init.c:0
	cancellation.c:81		memset.S:1133		Geometry.h:0
	stl_vector.h:0		malloc.c:0		Remaining MPI Time

Case Study: Identify Slow-Down Causes

Zoom into specific benchmarks (**AMG2013**) and examine the children of specific MPI functions (**MPI_Allgather**)



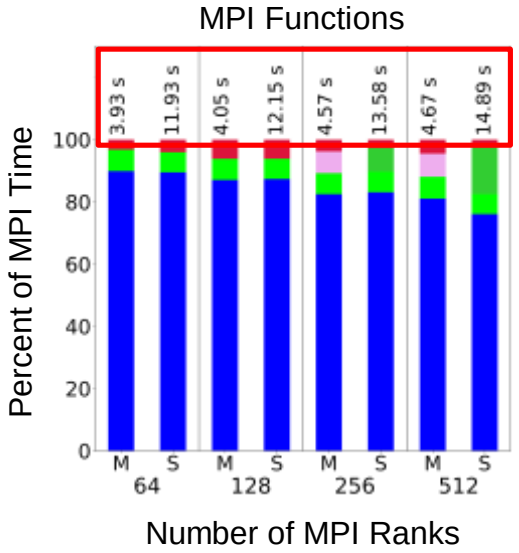


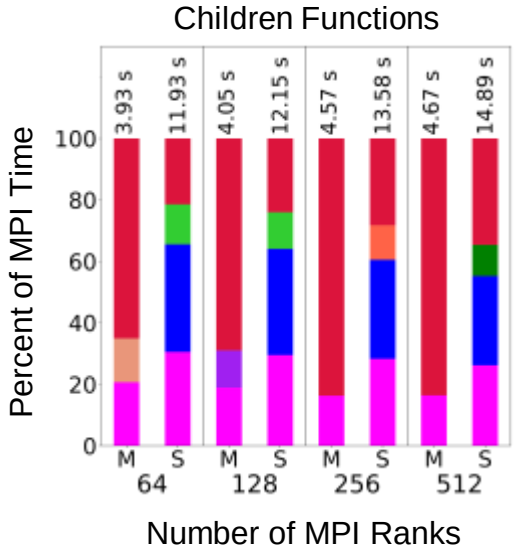
MPI Function Calls			
	MPI_Finalize		MPI_Send
	MPI_Allreduce		MPI_Wait
	MPI_Allgather		MPI_Waitany
	MPI_Waitall		MPI_Alltoallv
	MPI_Testany		Remaining MPI Time

Child Function Calls					
	<unknown file> [libopen-pal.so.3.1.0]:0		<unknown file> [libmlx5.so.1.0.0]:1133		<unknown file> [libmlx5.so.1.0.0]:0
	<unknown file> [libpami.so.3.1.0]:0		pthread_spin_lock.c:26		syscall-template.S:81
	syscall-template.S:82		pml_pami_send.c:0		pml_pami_init.c:0
	cancellation.c:81		memset.S:1133		Geometry.h:0
	stl_vector.h:0		malloc.c:0		Remaining MPI Time

Case Study: Identify Slow-Down Causes

Zoom into specific benchmarks (**AMG2013**) and examine the children of specific MPI functions (**MPI_Allgather**)

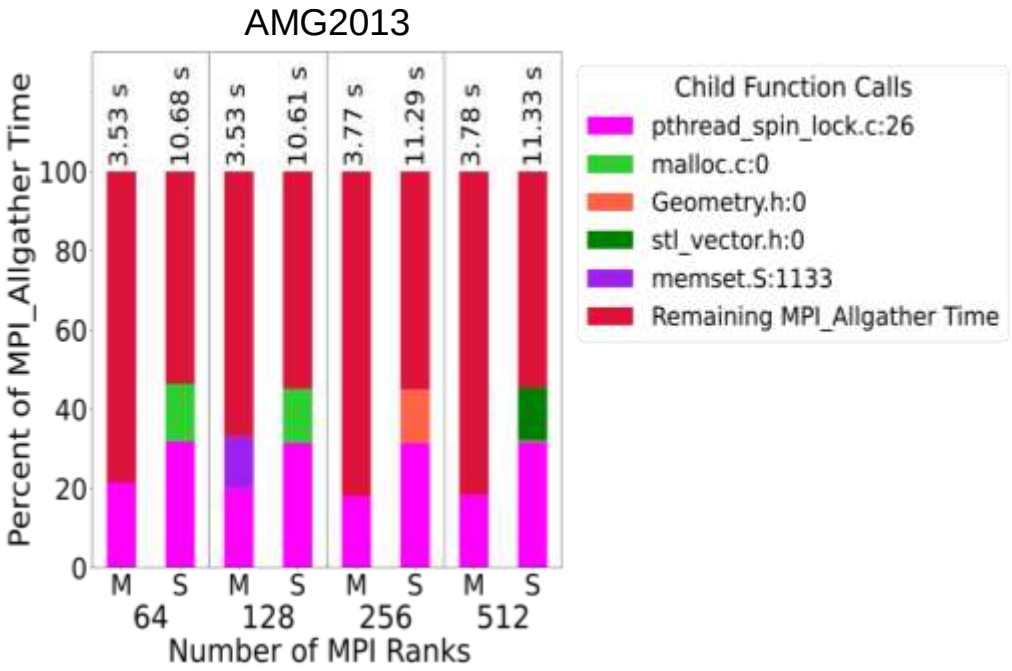




MPI Function Calls			
	MPI_Finalize		MPI_Send
	MPI_Allreduce		MPI_Wait
	MPI_Allgather		MPI_Waitany
	MPI_Waitall		MPI_Alltoallv
	MPI_Testany		Remaining MPI Time

Child Function Calls					
	<unknown file> [libopen-pal.so.3.1.0]:0		<unknown file> [libmlx5.so.1.0.0]:1133		<unknown file> [libmlx5.so.1.0.0]:0
	<unknown file> [libpami.so.3.1.0]:0		pthread_spin_lock.c:26		syscall-template.S:81
	syscall-template.S:82		pml_pami_send.c:0		pml_pami_init.c:0
	cancellation.c:81		memset.S:1133		Geometry.h:0
	stl_vector.h:0		malloc.c:0		Remaining MPI Time

Case Study: Identify Slow-Down Causes



Lessons Learned: MPI

- In our tests with MVAPICH and Spectrum-MPI, we learn that:
  - The **pthread_spin_lock** function is consistently a major contributor to MPI runtime (i.e., 10% or more of MPI time, usually 20% or more)
  - In AMG2013, the worse performance of **MPI_Allgather** in Spectrum-MPI can possibly be attributed to the use of **pthread_spin_lock**

Lessons Learned: Hatchet

- Using the query language and Hatchet, we are able to:
  - Extract all call paths specific to a given library
  - Determine the performance contributions of function calls used by these libraries
  - Correlate children function calls to specific important library API calls in an application
  - Use this correlation to determine children function calls that contribute the most to the performance of the targeted library API call
  - Compare the correlation of children and API calls across libraries to determine possible causes for performance differences

Future Work

- Use and expand on the techniques and tools from this work to study large scale scalability and replicability problems
  - Use information gained from studying these problems to propose mitigation strategies
  - Develop tools to help others study and mitigate these problems

Supplemental Slides

Hatchet Query Language

- User Input:
  - A *Query Path* represented as a list of *abstract graph nodes*

Hatchet Query Language

- User Input:
  - A *Query Path* represented as a list of *abstract graph nodes*
    - Two parts to an *abstract graph node*:

Hatchet Query Language

- User Input:
  - A *Query Path* represented as a list of *abstract graph nodes*
    - Two parts to an *abstract graph node*:
      1. A wildcard specifying the number of real graph nodes to match to the *abstract graph node* (default is “.”)

Hatchet Query Language

- User Input:
  - A *Query Path* represented as a list of *abstract graph nodes*
    - Two parts to an *abstract graph node*:
      1. A wildcard specifying the number of real graph nodes to match to the *abstract graph node* (default is “.”)
      2. A filter used to determine whether a real graph node matches the *abstract graph node* (default is an “always true” filter)

Hatchet Query Language: APIs

Internal Representation

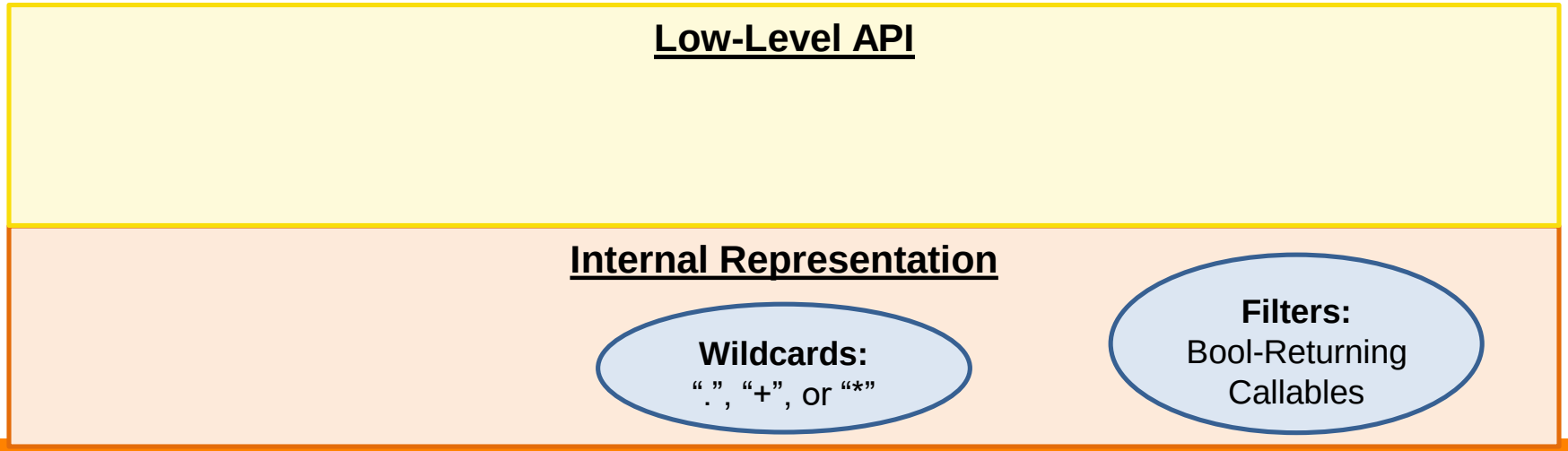
Hatchet Query Language: APIs

Internal Representation

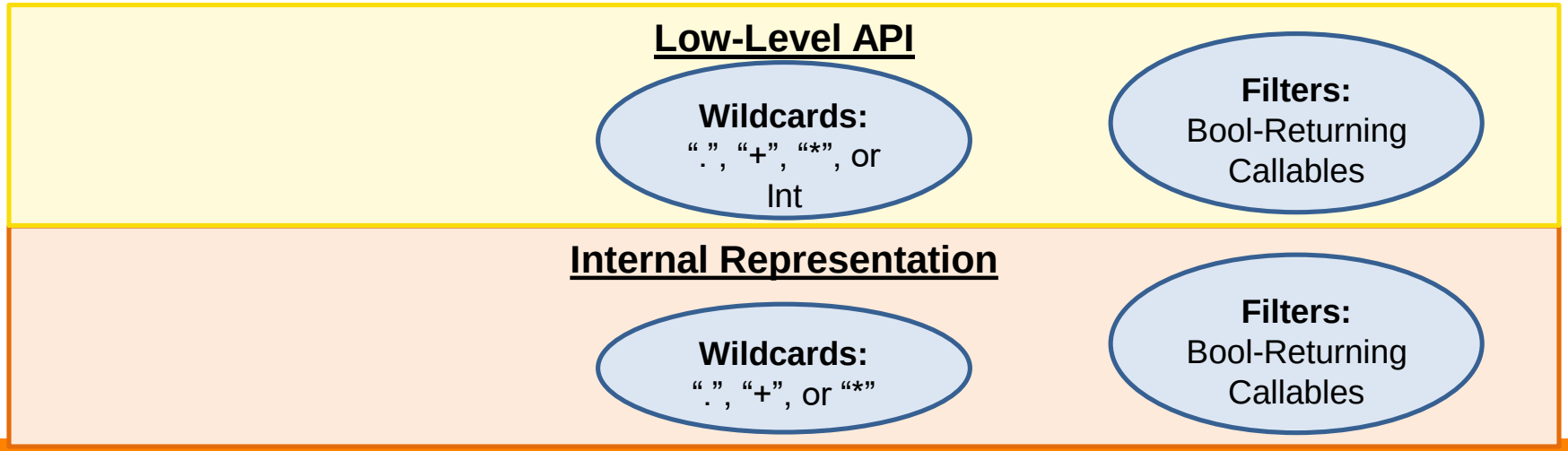
Wildcards:
“.”, “+”, or “*”

Filters:
Bool-Returning
Callables

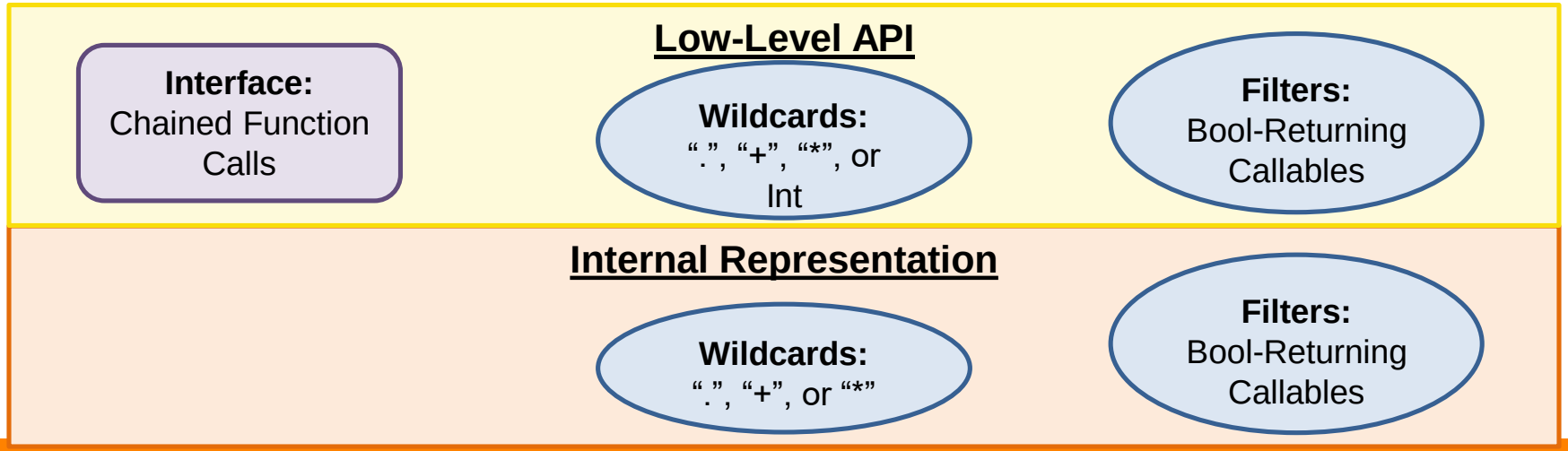
Hatchet Query Language: APIs



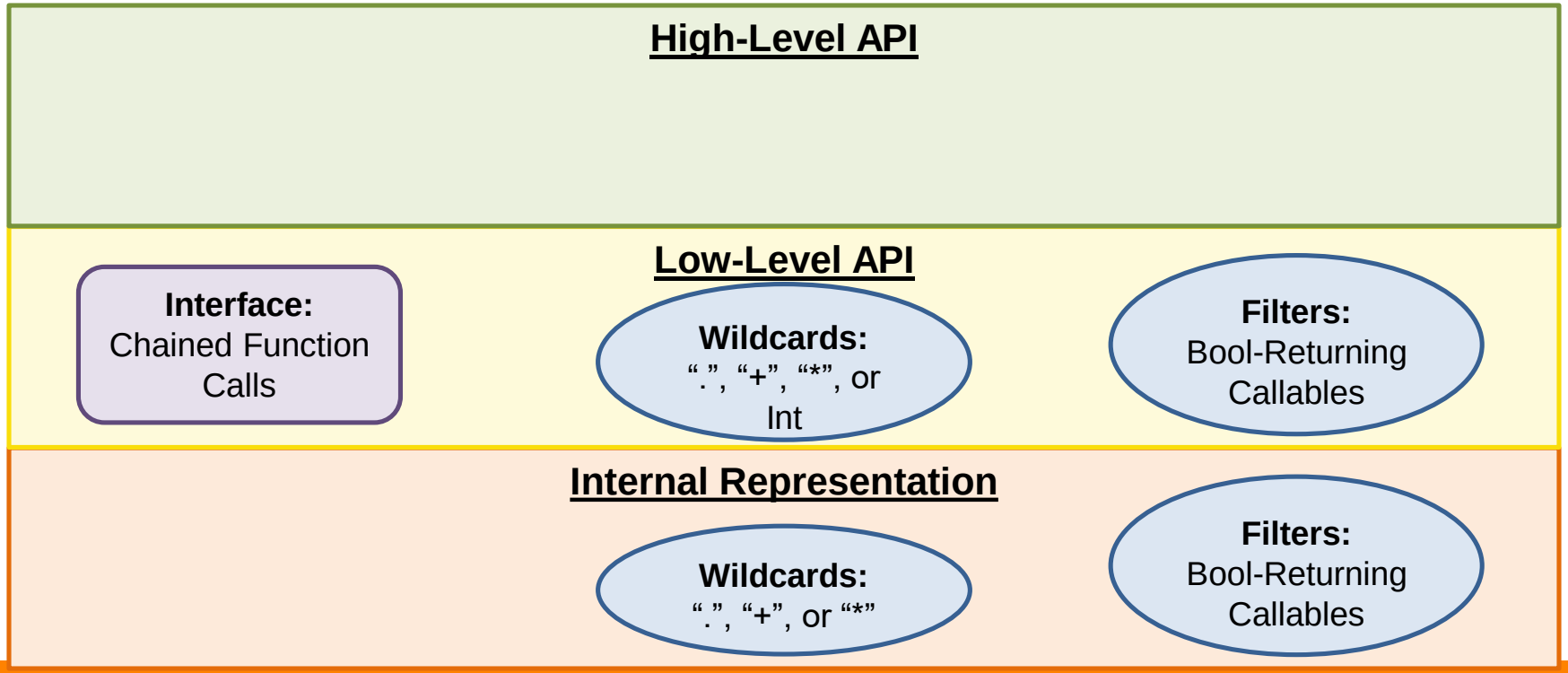
Hatchet Query Language: APIs



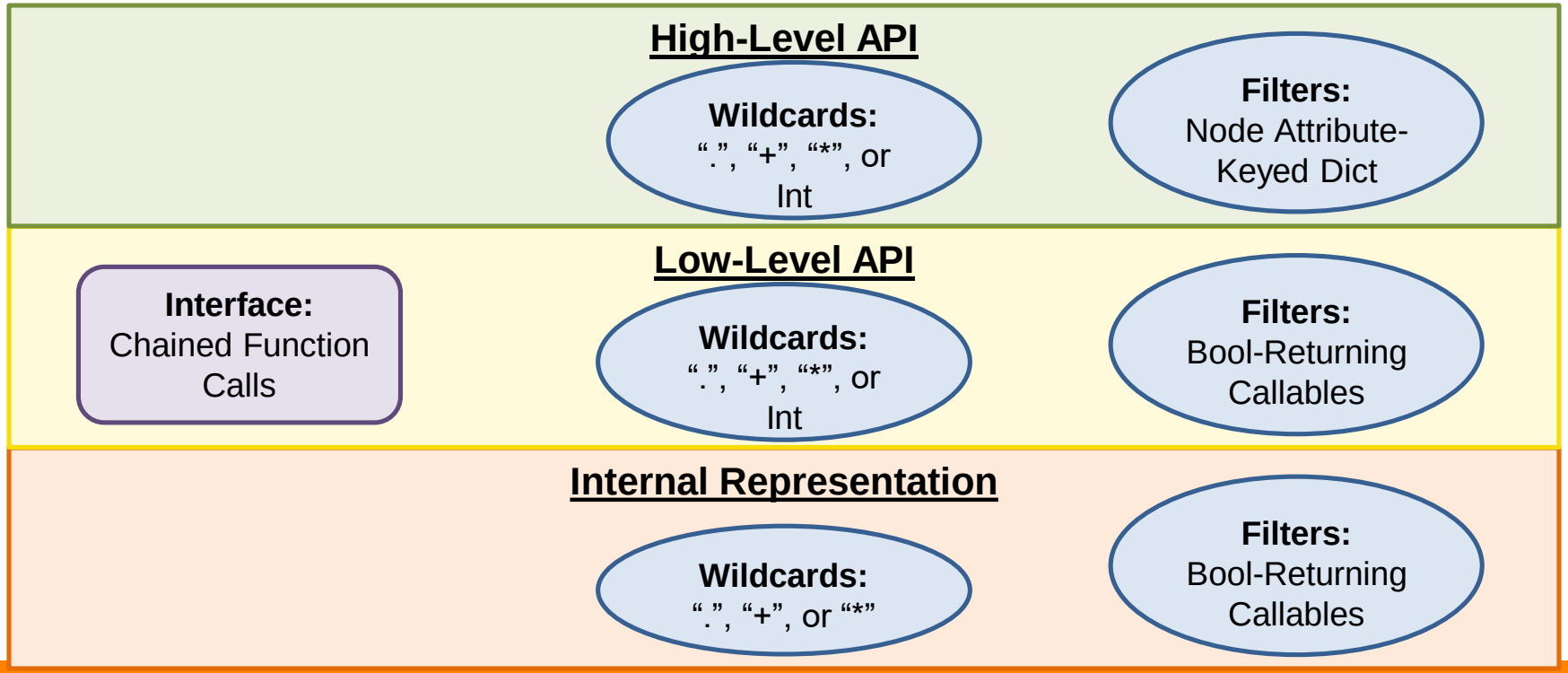
Hatchet Query Language: APIs



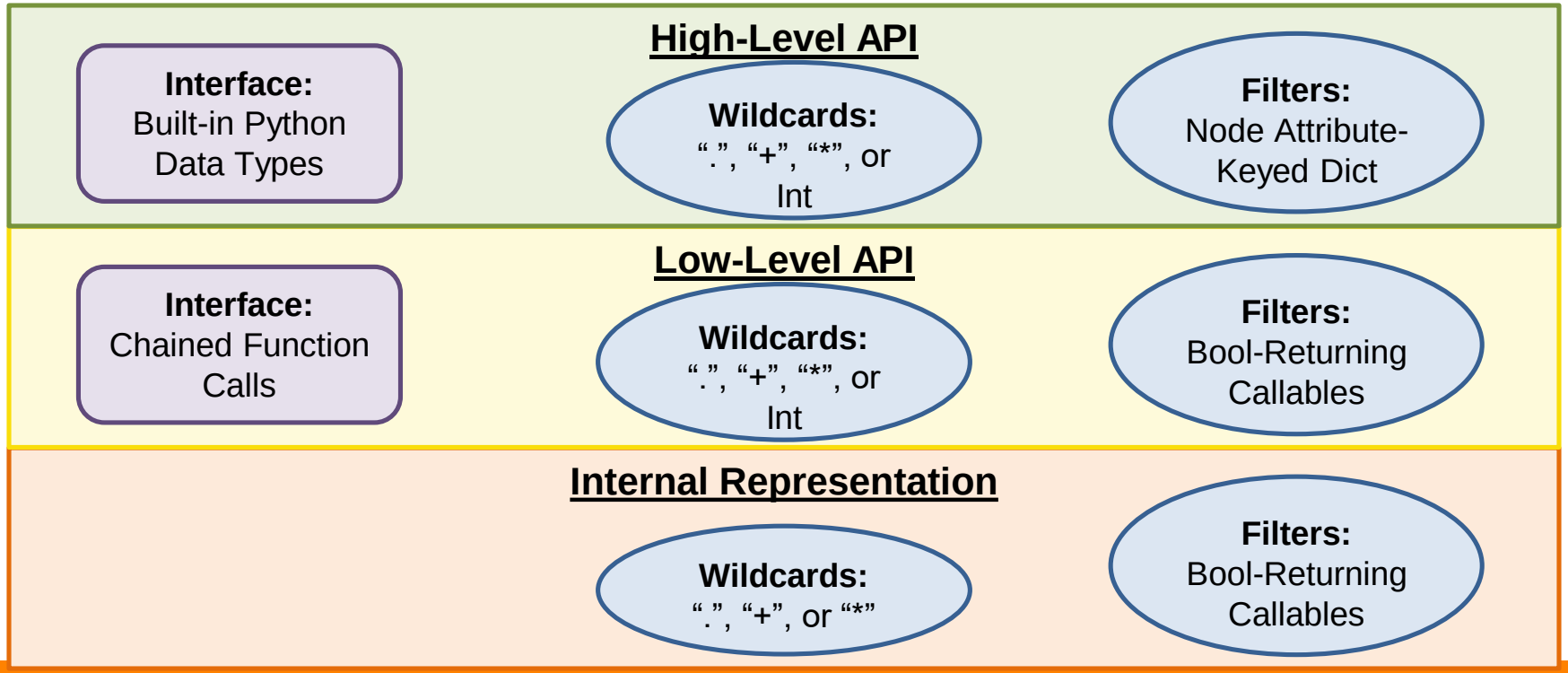
Hatchet Query Language: APIs



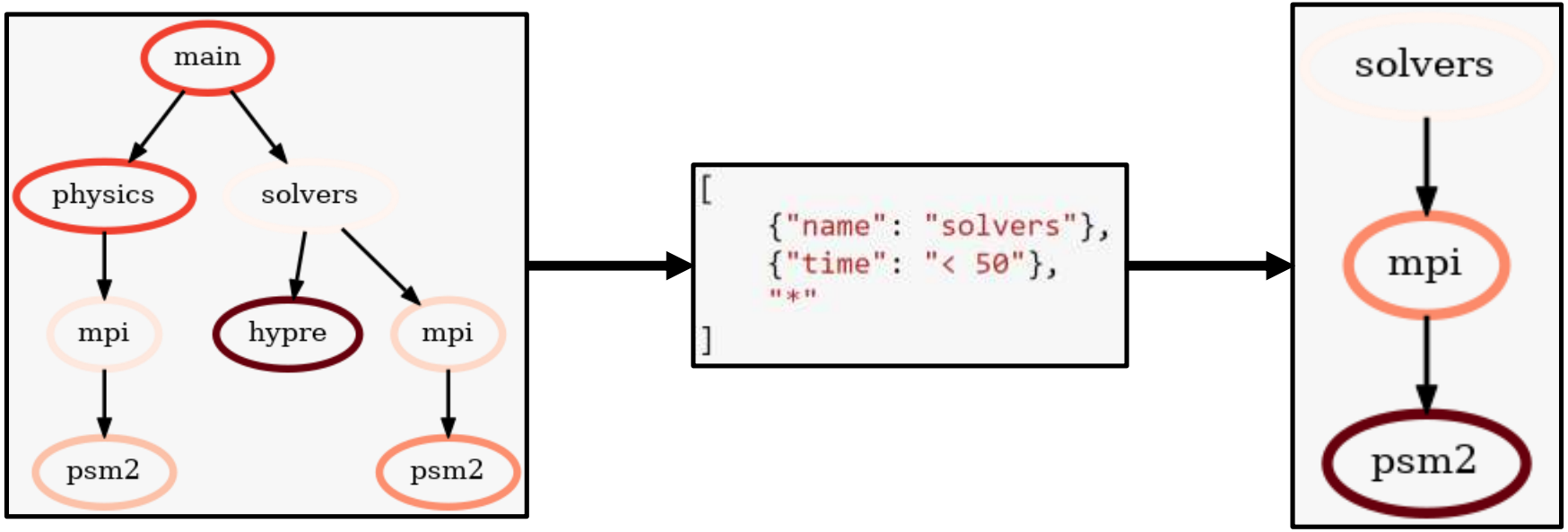
Hatchet Query Language: APIs



Hatchet Query Language: APIs



Hatchet Query Language: Example



Hatchet Query Language: Examples

```
def filter_percent_mpi_time(gf0, percent=0.1):

    def match_mpi(row):
        if (not row["name"].startswith("PMPI_") and
                not row["name"].startswith("MPI_") and
                not row["name"].startswith("MPIIO_")):
            return False
        curr_node = row.name
        for node in curr_node.traverse():
            if gf.dataframe.at[node, "comp_time"] < 0. and row.name != node:
                gf.dataframe.at[node, "comp_time"] = row["time (inc)"]
        return True

    def match_time_filter(row):
        if row["comp_time"] >= 0. and row["time"] >= percent*row["comp_time"]:
            return True
        return False

    gf = gf0.deepcopy()
    gf.dataframe["comp_time"] = -1.0
    print("Graph size before filter: {}".format(len(gf.graph)))
    query = (ht.QueryMatcher()
                .match("*")
                .rel(filter_func=match_mpi)
                .rel("*")
                .rel(filter_func=match_time_filter)
                .rel("*")
            )
    fgf = gf.filter(query, squash=True)
    print("Graph size after filter: {}".format(len(fgf.graph)))
    return fgf
```

- Gets all subpaths from roots to leaves containing the following:
  - An MPI Node
  - A node representing a child call of the MPI function that takes at least **percent** percent of the MPI function's total time

Case Study: Benchmarks

- Kripke (LLNL) [5]
  - Mini-App Benchmark
  - Deterministic
  - Runs S_N Neutron Transport simulation

[5] AJ Kunen et al. 2015. *KRIPKE-A massively parallel transport mini-app*. Technical Report. Lawrence Livermore National Laboratory (LLNL)

Case Study: Benchmarks

- Lammps (SNL) [6]
  - Popular classical molecular dynamics simulator
  - Also contains a set of benchmarks

[6] S. Plimpton. 1995. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comput. Phys.* 117, 1 (March 1995), 1-19

Case Study: Benchmarks

- AMG2013 (LLNL) [7]
  - Parallel algebraic multigrid solver for linear systems
  - Proxy App for hypre's BoomerAMG solver
  - Highly synchronous code

[7] U. Yang et al. 2017. Algebraic Multigrid Benchmark, Version 00. (8 2017)

Case Study: MPIs

- MVAPICH (v2.3.3)
  - Project led by the Network-Based Computing Laboratory at Ohio State University
  - Based on MPICH (Argonne National Lab)
  - Conforms with MPI 3.1 standard

Case Study: MPIs

- Spectrum-MPI (v10.2)
  - Created by IBM
  - Based on OpenMPI (first created by LANL, Indiana University, and the University of Tennessee)
  - Conforms with MPI 3.1 standard