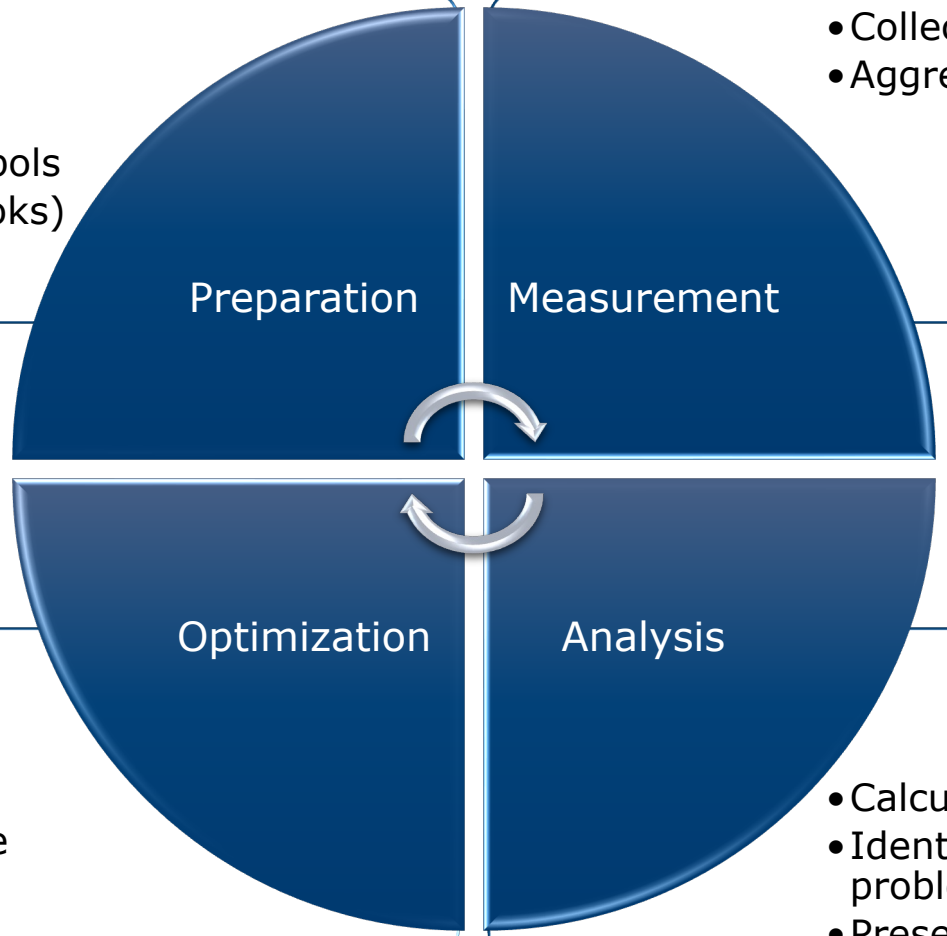
International HPC Summer School 2023: Performance analysis and optimization

Tools Overview

Ilya Zhukov
Jülich Supercomputing Centre

Performance engineering workflow

- Build model of predicted performance
- Select data to measure
- Prepare application with symbols
- Insert extra code (probes/hooks)



- Collection of performance data
- Aggregation of performance data

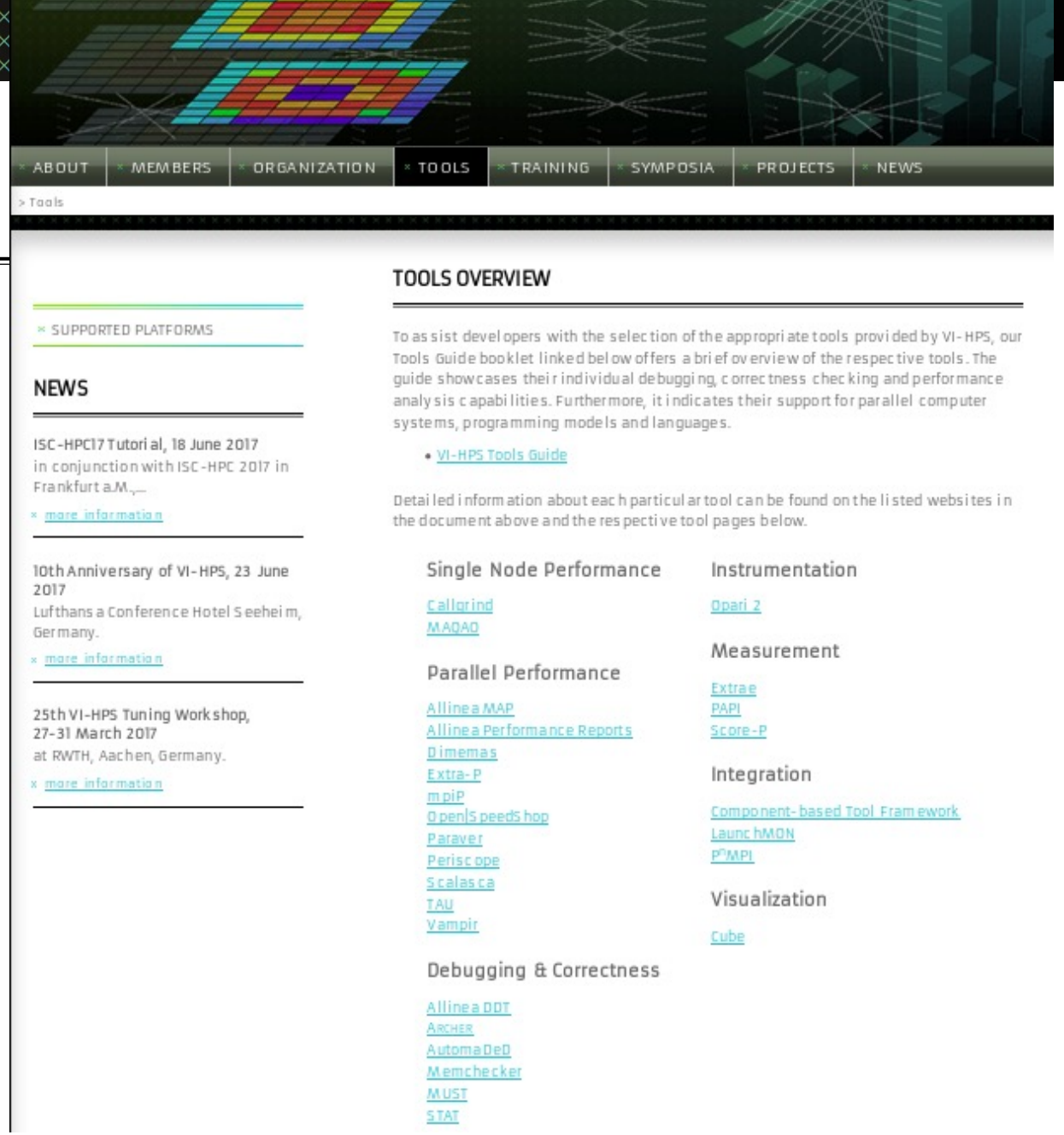
- Modifications intended to eliminate/reduce performance problem

- Calculation of metrics
- Identification of performance problems
- Presentation of results

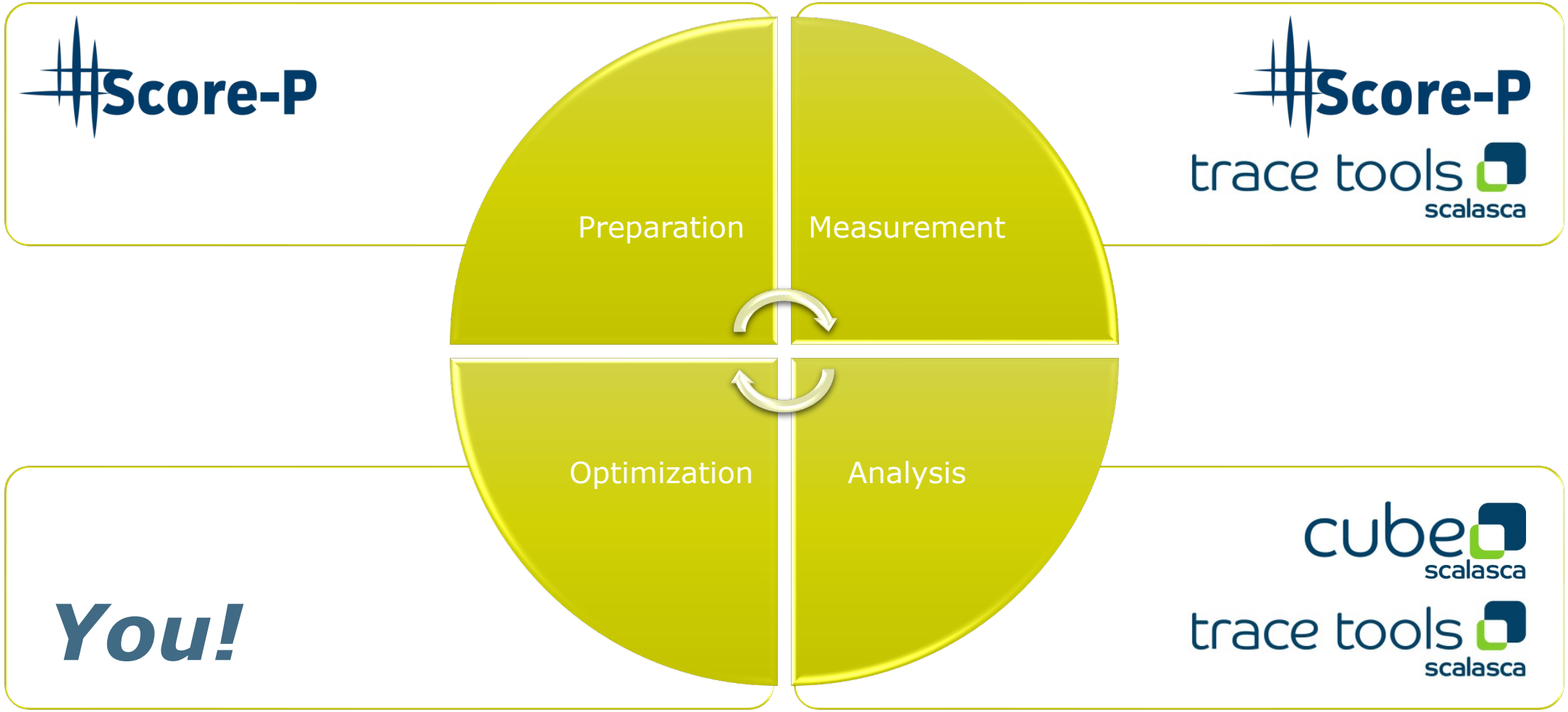
Which tools to use?

- Overview at http://www.vi-hps.org/tools/
  - Not complete though, HPCToolkit, PerfExpert, Intel Tools, Cray tools, likwid, … missing, no members off VI-HPS
  - More than just performance tools
  - 2-page description of tools: VI-HPS Tools Guide

- For this session we use:
  - Score-P for measurement
  - Scalasca
  - Cube
  - Vampir



Performance engineering workflow



Score-P instrumentation and measurement infrastructure

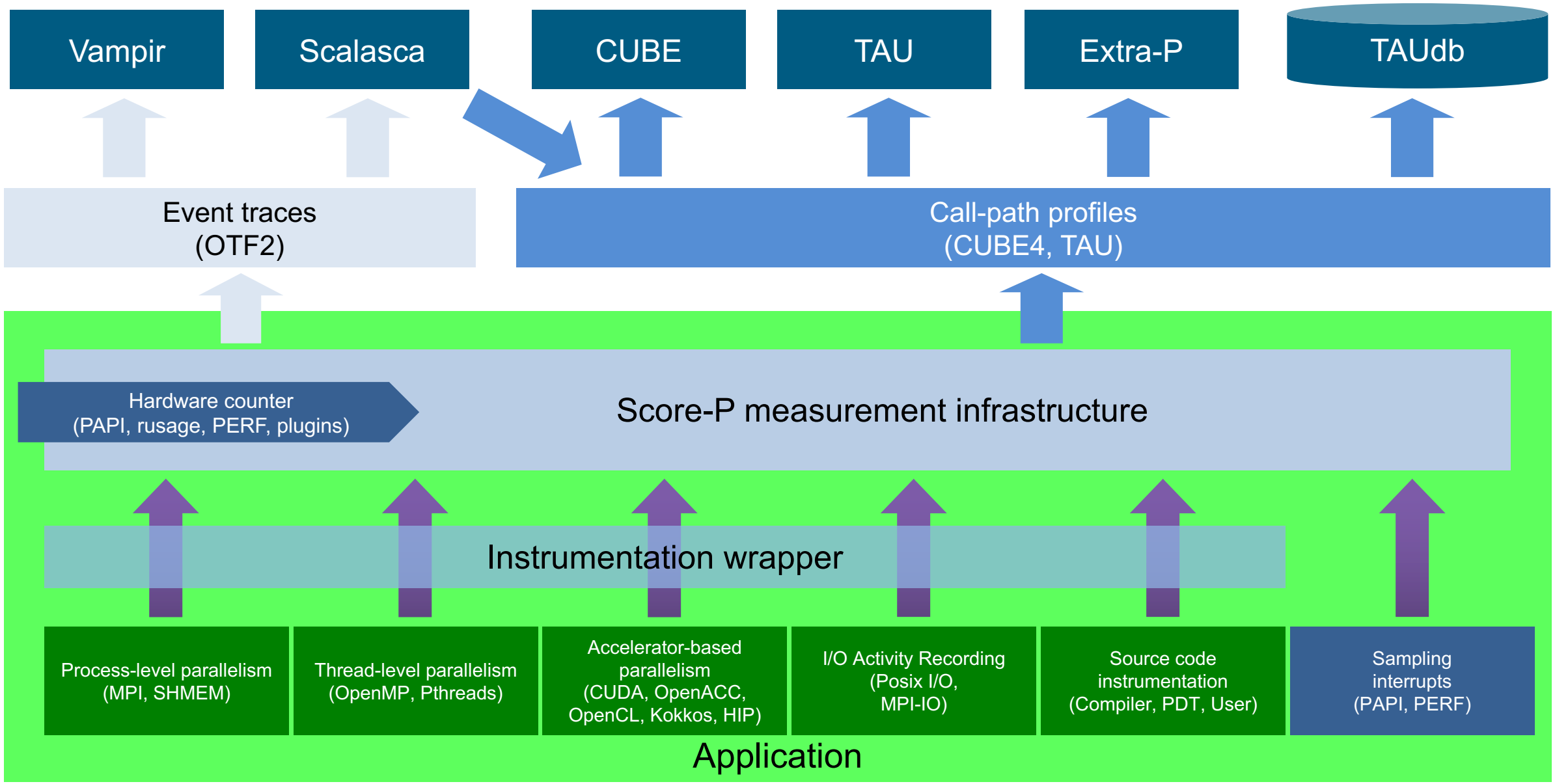
Score-P

DOI 10.5281/zenodo.1240731

- Infrastructure for instrumentation and performance measurements
- Instrumented application can be used to produce several results:
  - Call-path profiling: CUBE4 data format used for data exchange
  - Event-based tracing: OTF2 data format used for data exchange
- Supported parallel paradigms:
  - Multi-process: MPI, SHMEM
  - Thread-parallel: OpenMP, POSIX threads
  - Accelerator-based: CUDA, HIP, OpenCL, OpenACC

- Initial project funded by BMBF
- Close collaboration with PRIMA project funded by DOE

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung


Vampir
Scalasca
CUBE
TAU
Extra-P
TAUdb
Event traces
(OTF2)
Call-path profiles
(CUBE4, TAU)
Hardware counter
(PAPI, rusage, PERF, plugins)
Score-P measurement infrastructure
Instrumentation wrapper
Process-level parallelism
(MPI, SHMEM)
Thread-level parallelism
(OpenMP, Pthreads)
Accelerator-based
parallelism
(CUDA, OpenACC,
OpenCL, Kokkos, HIP)
I/O Activity Recording
(Posix I/O,
MPI-IO)
Source code
instrumentation
(Compiler, PDT, User)
Sampling
interrupts
(PAPI, PERF)
Application

Score-P features

- Open source: 3-clause BSD license
  - Commitment to joint long-term cooperation
  - Development based on meritocratic governance model
  - Open for contributions and new partners
- Portability: supports all major HPC platforms
- Scalability: successful measurements with >1M threads
- Functionality:
  - Generation of call-path profiles and event traces (supporting highly scalable I/O)
  - Using direct instrumentation and sampling
  - Flexible measurement configuration without re-compilation
  - Recording of time, visits, communication data, hardware counters
  - Support for MPI, SHMEM, OpenMP, Pthreads, CUDA, HIP, OpenCL, OpenACC and valid combinations
- Latest release: Score-P 8.1 (April 2023)

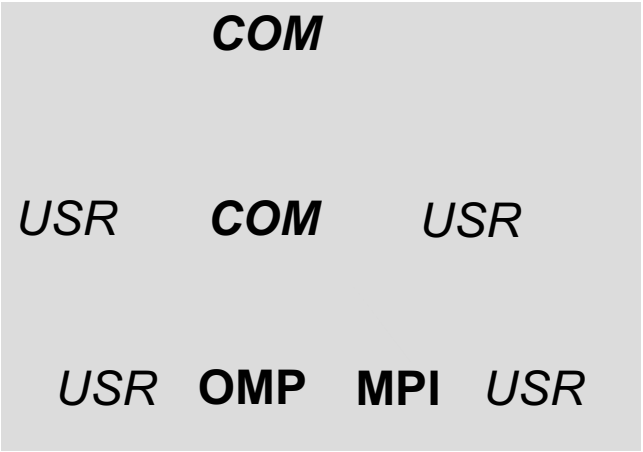
Call-path Profile: Recipe

1. Prefix your ***compile & link commands*** with
   "`scorep`"
2. Prefix your MPI ***launch command*** with
   "`scalasca -analyze`"
3. After execution, compare overall runtime with uninstrumented run to determine overhead
4. If overhead is too high
   1. Score measurement using
      "`scalasca -examine -s scorep_<title>`"
   2. Prepare filter file
   3. Re-run measurement with filter applied using prefix
      "`scalasca –analyze –f <filter_file>`"
5. After execution, examine analysis results using
   "`scalasca -examine scorep_<title>`"

Call-path Profile: Example (cont.)

```
% scalasca -examine -s scorep_myprog_Ppnxt_sum
scorep-score -r ./scorep_myprog_Ppnxt_sum/profile.cubex
INFO: Score report written to ./scorep_myprog_Ppnxt_sum/scorep.score
```

- Estimates trace buffer requirements
- Helps to identify candidate functions for filtering
  - Computational routines with high visit count and low time-per-visit ratio
- Region/call-path classification
  - MPI (pure MPI library functions)
  - OMP (pure OpenMP functions/regions)
  - USR (user-level source local computation
  - COM ("combined" USR + OpenMP/MPI)
  - ANY/ALL (aggregate of all region types)



Call-path Profile: Example (cont.)

```
% less scorep_myprog_Ppmxt_sum/scorep.score
Estimated aggregate size of event trace:                   162GB
Estimated requirements for largest trace buffer (max_buf): 2758MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       2822MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=2822MB to avoid
 intermediate flushes or reduce requirements using USR regions filters.)

flt type    max_buf[B]        visits  time[s] time[%] time/     region
                                                    visit[us]
     ALL 2,891,417,902 6,662,521,083 36581.51   100.0      5.49  ALL
     USR 2,858,189,854 6,574,882,113 13618.14    37.2      2.07  USR
     OMP    54,327,600    86,353,920 22719.78    62.1    263.10  OMP
     MPI       676,342       550,010   208.98     0.6    379.96  MPI
     COM       371,930       735,040    34.61     0.1     47.09  COM

     USR   921,918,660 2,110,313,472  3290.11     9.0      1.56  matmul_sub
     USR   921,918,660 2,110,313,472  5914.98    16.2      2.80  binvcrhs
     USR   921,918,660 2,110,313,472  3822.64    10.4      1.81  matvec_sub
     USR    41,071,134    87,475,200   358.56     1.0      4.10  lhsinit
     USR    41,071,134    87,475,200   145.42     0.4      1.66  binvrhs
     USR    29,194,256    68,892,672    86.15     0.2      1.25  exact_solution
     OMP     3,280,320     3,293,184    15.81     0.0      4.80  !$omp parallel
     [...]
```

Call-path Profile: Filtering

- In this example, the 6 most frequently called routines are
- of type USR
- These routines contribute around 35% of total time
  - However, much of that is most likely measurement overhead
    - Frequently executed
    - Time-per-visit ratio in the order of a few microseconds
- Avoid measurements of these to reduce the overhead
- List routines to be filtered in simple text file
- Specify SCOREP_FILTERING_FILE for measurement
  - or with GCC specify as --instrument-filter when instrumenting

Filtering: Example

```
% cat filter.txt
SCOREP_REGION_NAMES_BEGIN
    EXCLUDE
        binvcrhs
        matmul_sub
        matvec_sub
        binvrhs
        lhsinit
        exact_solution
SCOREP_REGION_NAMES_END
```

- Score-P filtering files support
  - Wildcards (shell globs)
  - Blacklisting
  - Whitelisting
  - Filtering based on filenames

Score-P: Advanced Features

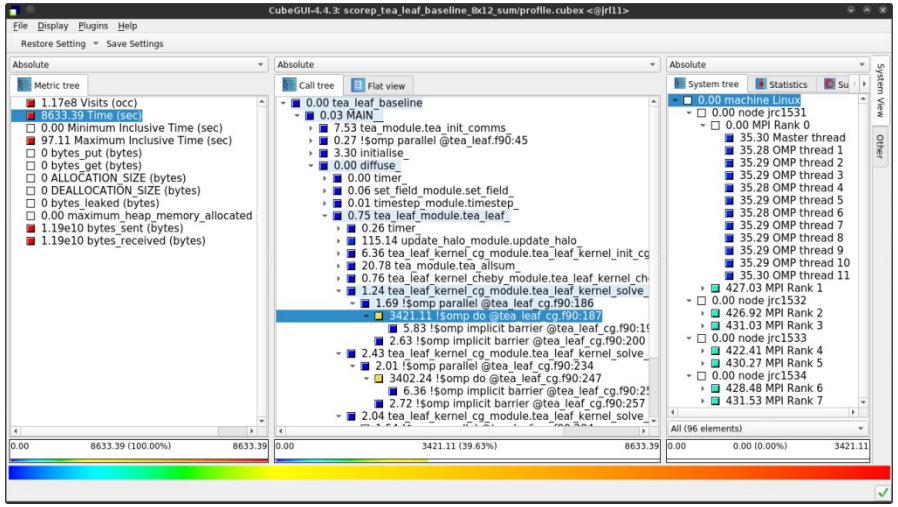
- Measurement can be extensively configured via environment variables
  - Check output of "`scorep-info config-vars`" for details
- Allows for targeted measurements:
  - Selective recording
  - Phase profiling
  - Parameter-based profiling
  - …
- Ask us or see the user manual for details

Cube performance report explorer

Cube

CubeLib DOI 10.5281/zenodo.1248078
CubeGUI DOI 10.5281/zenodo.1248087

- Parallel program analysis report exploration tools
  - Libraries for XML+binary report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
    - Requires Qt ≥ 5



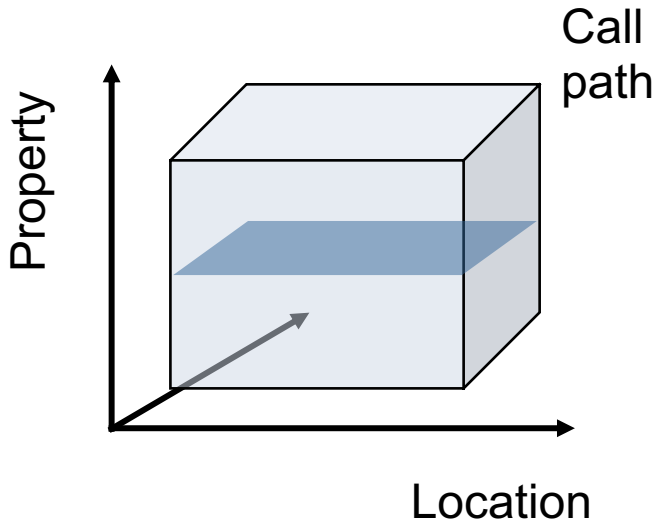
- Originally developed as part of the Scalasca toolset

- Now available as separate components
  - Can be installed independently of Score-P and Scalasca, e.g., on laptop or desktop
  - Latest releases: Cube v4.8.1 (March 2023)

Note: source distribution tarballs for Linux, as well as binary packages provided for Windows & MacOS, from **www.scalasca.org** website in Software/Cube 4.x

Analysis presentation and exploration

- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call path (program location)
  - System location (process/thread)

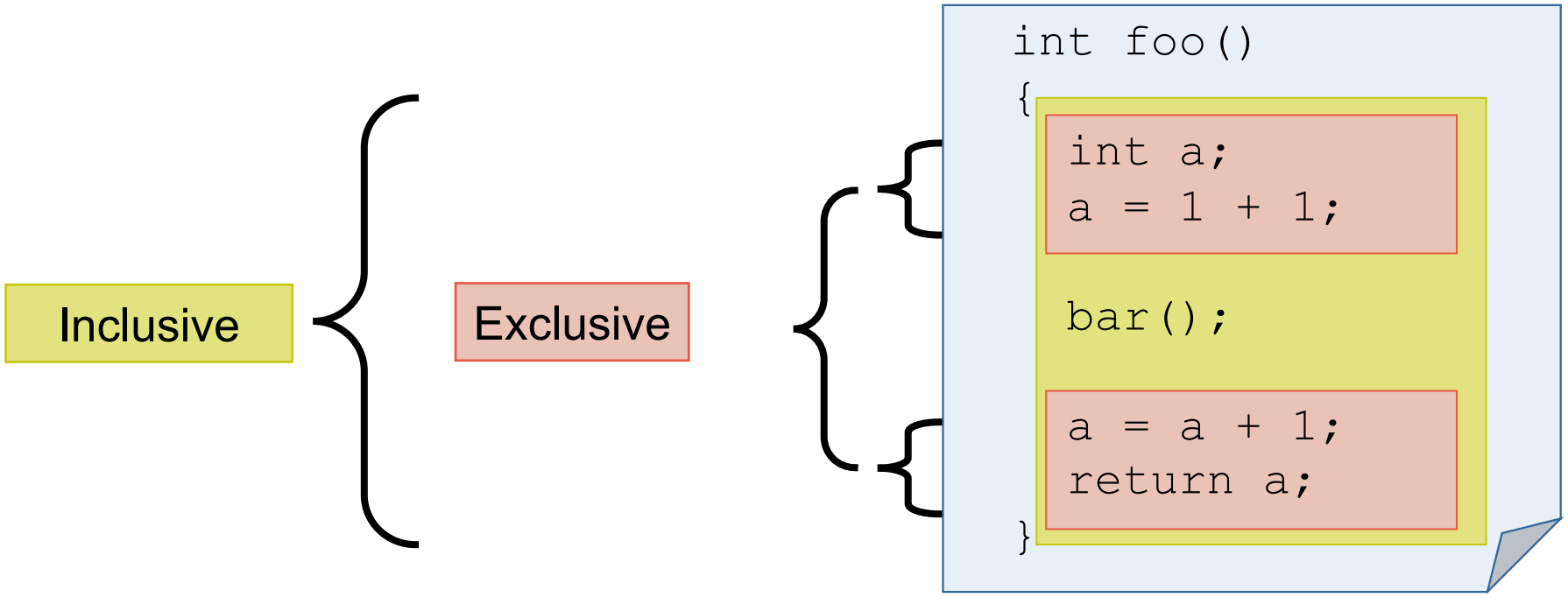


- Three coupled tree browsers

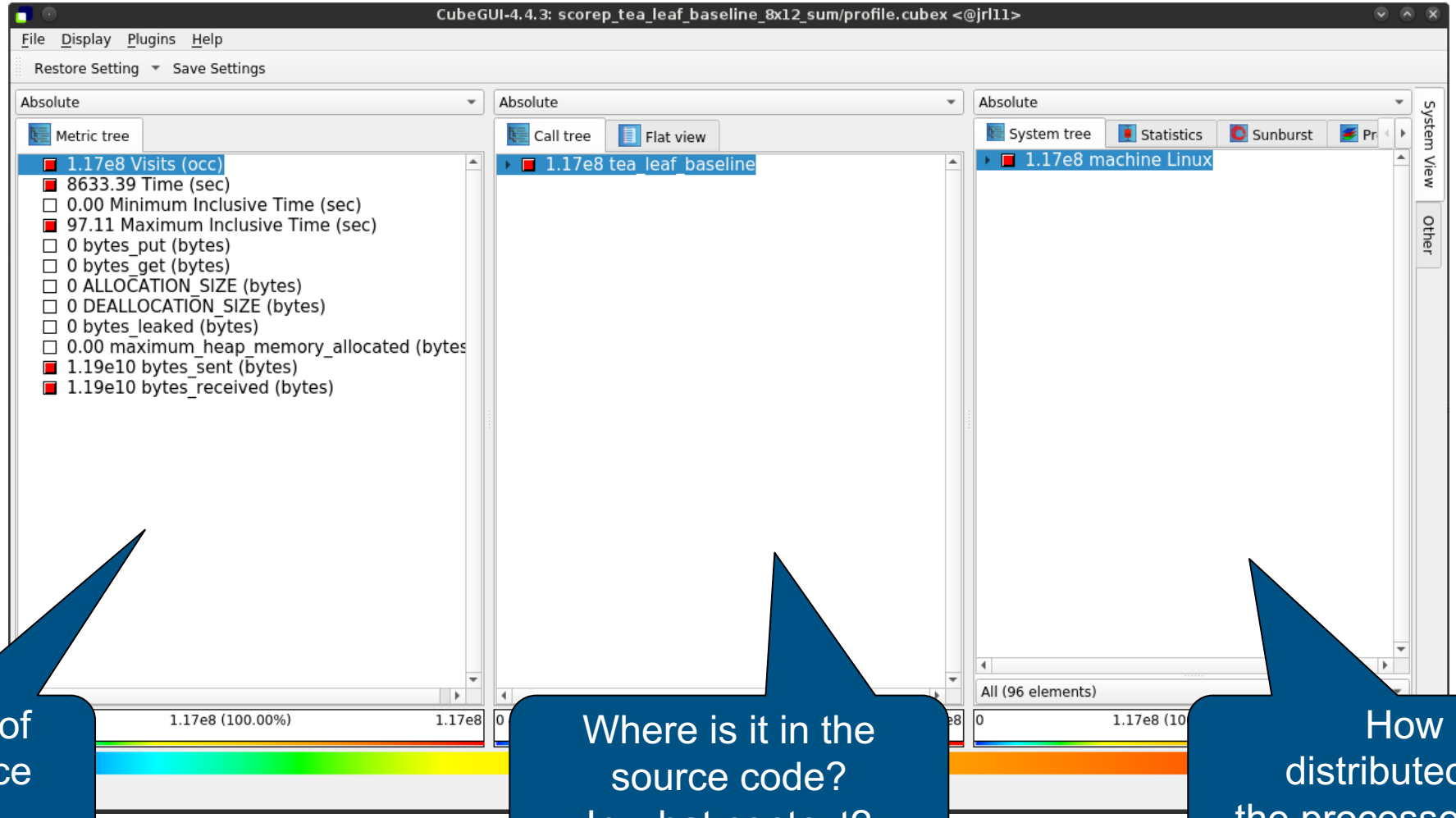
- Cube displays severities
  - As value: for precise comparison
  - As color: for easy identification of hotspots
  - Inclusive value when closed & exclusive value when expanded
  - Customizable via display modes

Inclusive vs. exclusive values

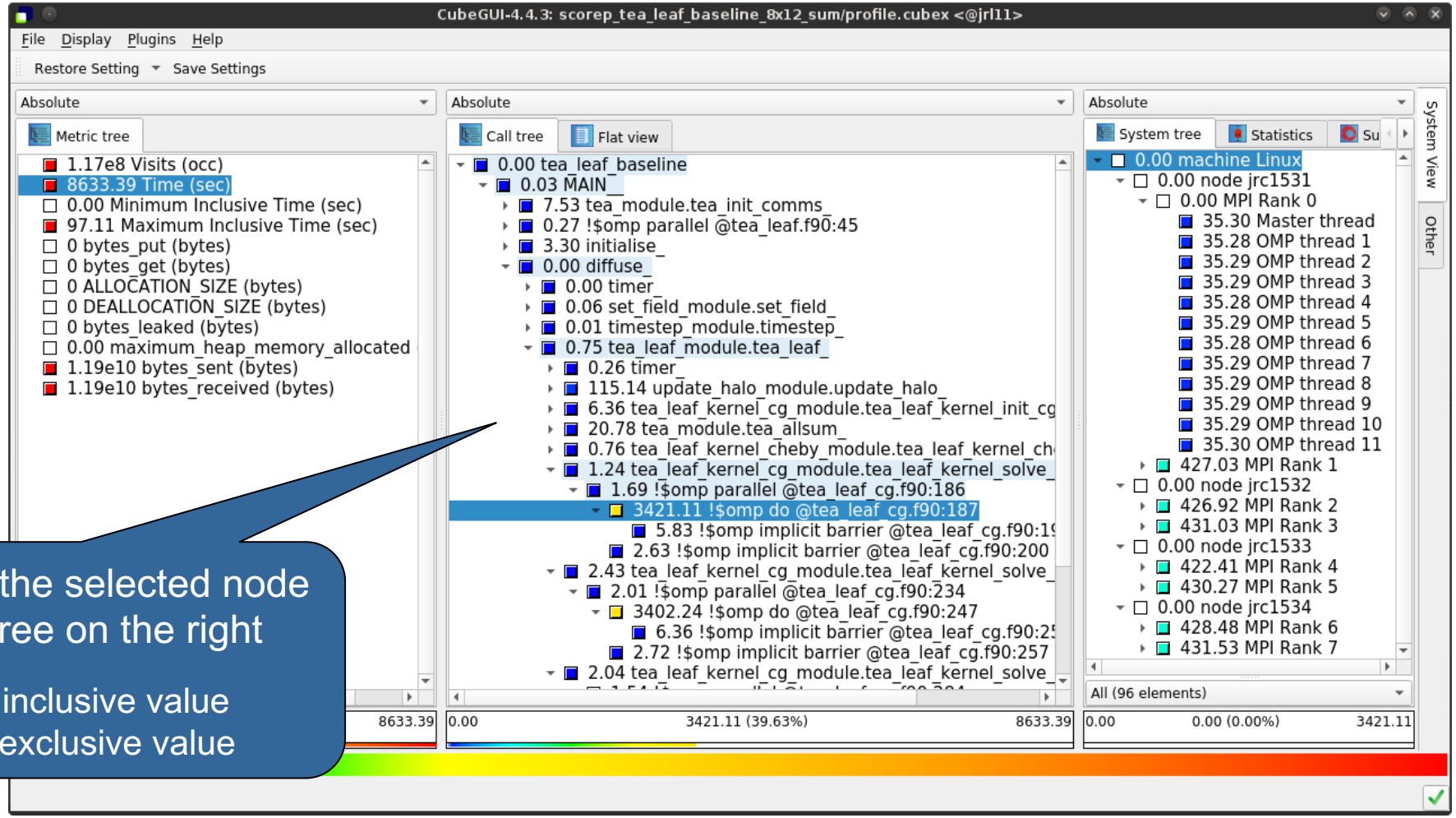
- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further



Plain summary analysis report (opening view)



Plain summary analysis report (expanded call tree/system tree)

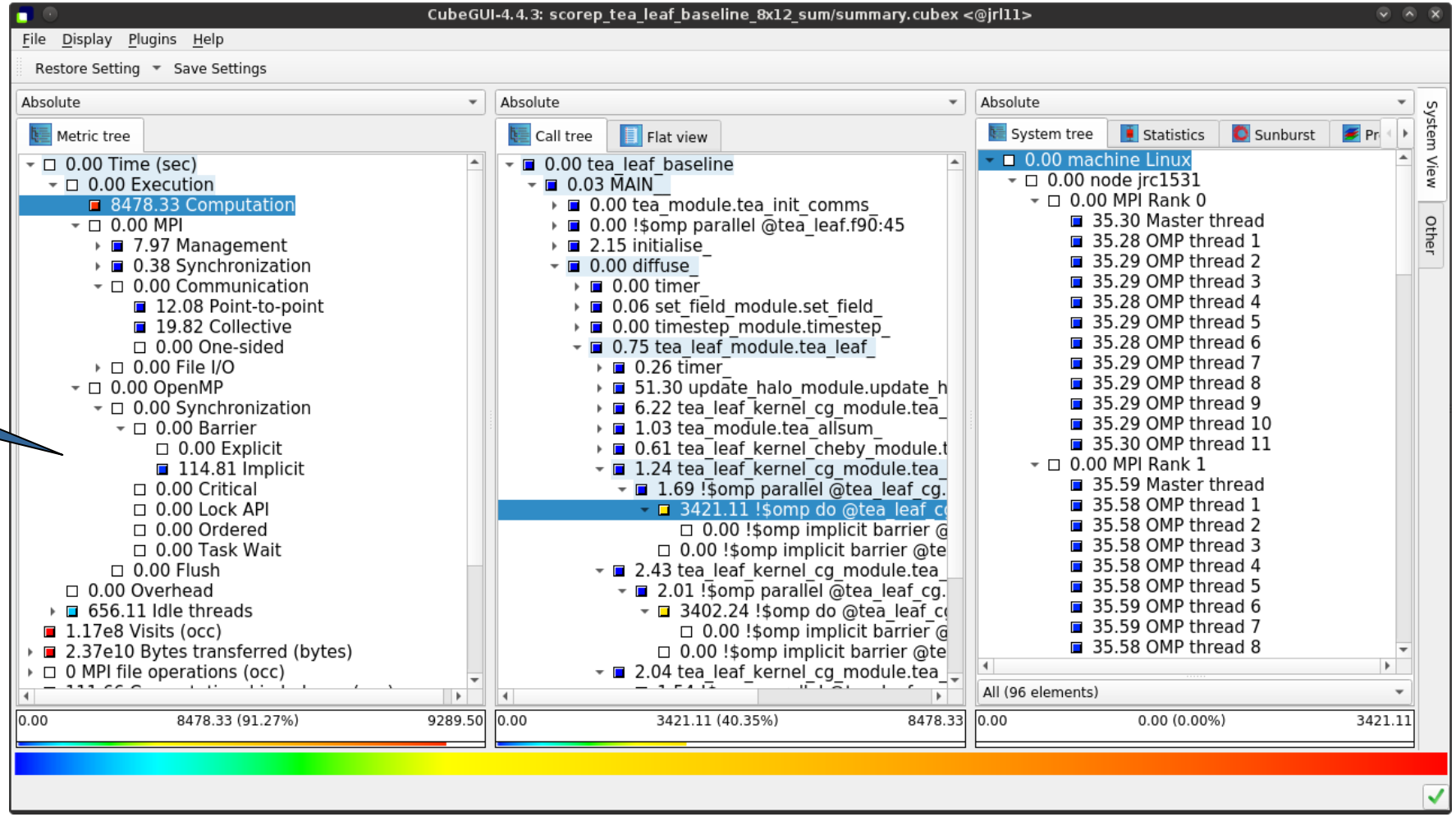


Distribution of the selected node across the tree on the right

Collapsed: inclusive value
Expanded: exclusive value

Post-processed summary analysis report (Scalasca)

Split base metrics from plain report into hierarchy of more specific metrics



Scalasca Trace Tools

Scalasca Trace Tools

DOI 10.5281/zenodo.4103922

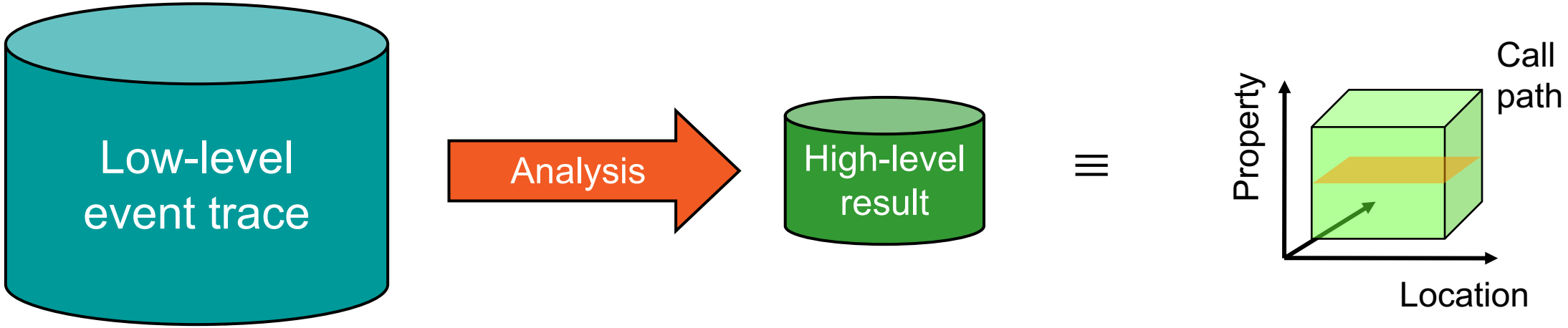
- **Scalable trace-based** performance analysis toolset for the most popular parallel programming paradigms
  - Current focus: MPI, OpenMP, and (to a limited extend) POSIX threads
  - Analysis of traces including only host-side events from applications using CUDA, OpenCL, or OpenACC (also in combination with MPI and/or OpenMP) is possible, but results need to be interpreted with some care

- Specifically targeting large-scale parallel applications
  - Demonstrated scalability up to 1.8 million parallel threads
  - Of course also works at small/medium scale

- Latest release:
  - Scalasca Trace Tools v2.6.1 (December 2022), coordinated with Score-P v8.0

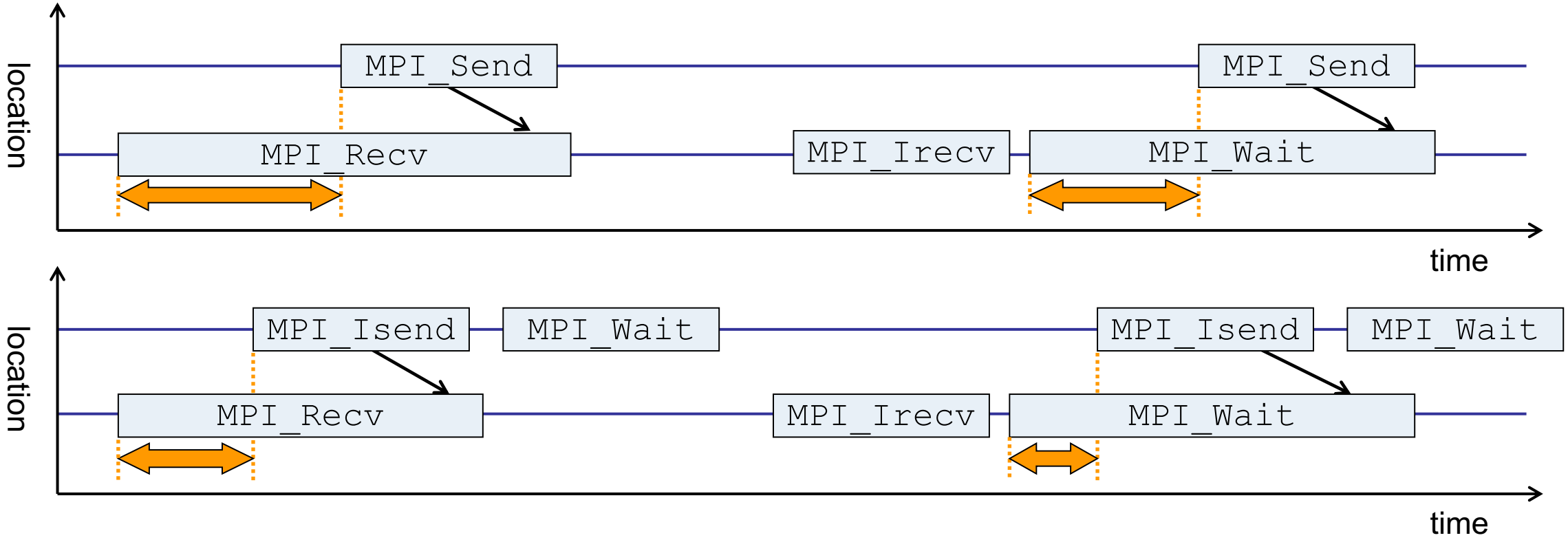
Automatic trace analysis

- Idea
  - Automatic search for patterns of inefficient behavior
  - Classification of behavior & quantification of significance
  - Identification of delays as root causes of inefficiencies



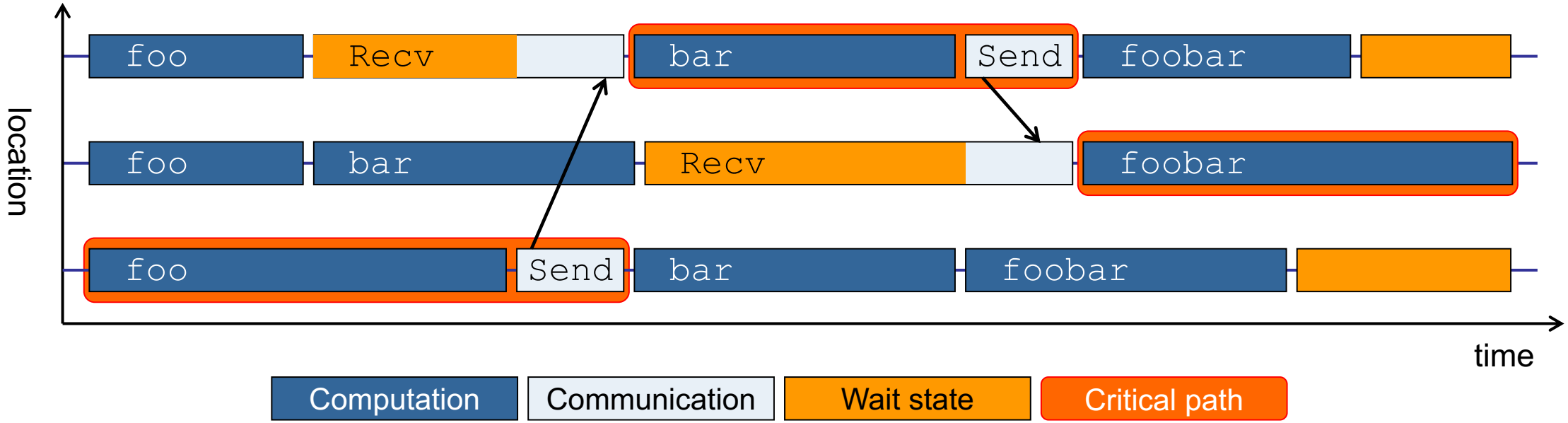
  - Guaranteed to cover the entire event trace
  - Quicker than manual/visual trace analysis
  - Parallel replay analysis exploits available memory & processors to deliver scalability

Example: "*Late Sender*" wait state



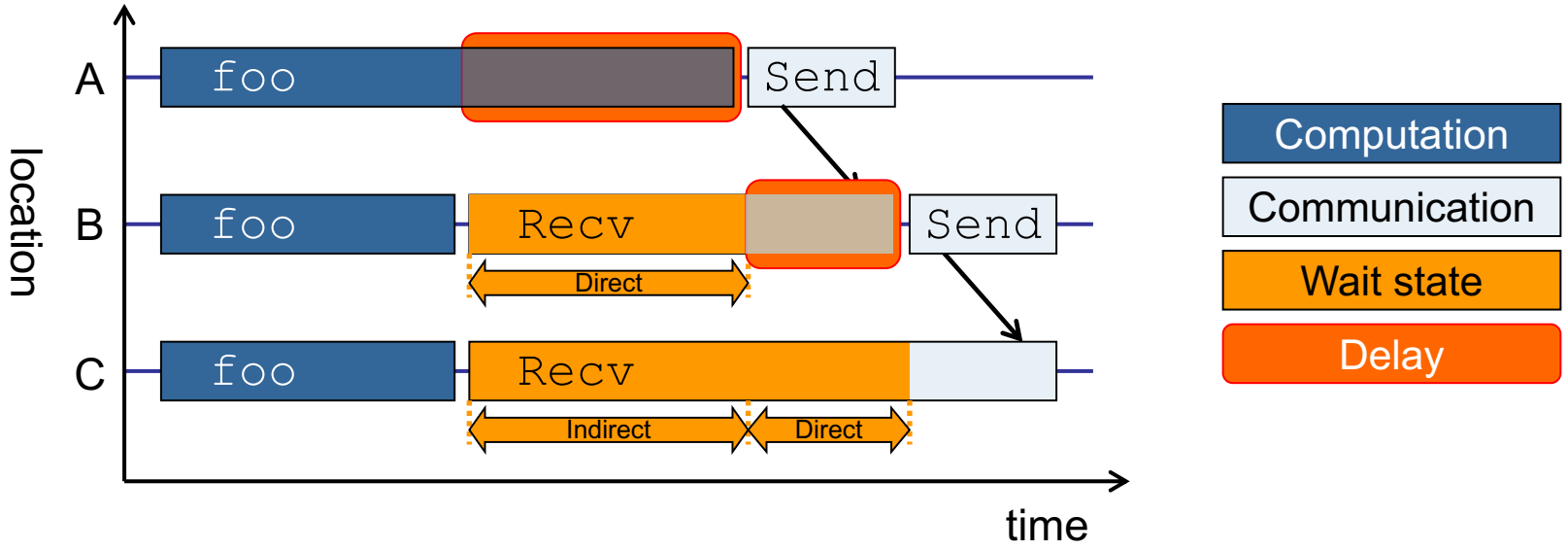
- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication

Example: Critical path



- Shows call paths and processes/threads that are responsible for the program's wall-clock runtime
- Identifies good optimization candidates and parallelization bottlenecks

Example: Root-cause analysis



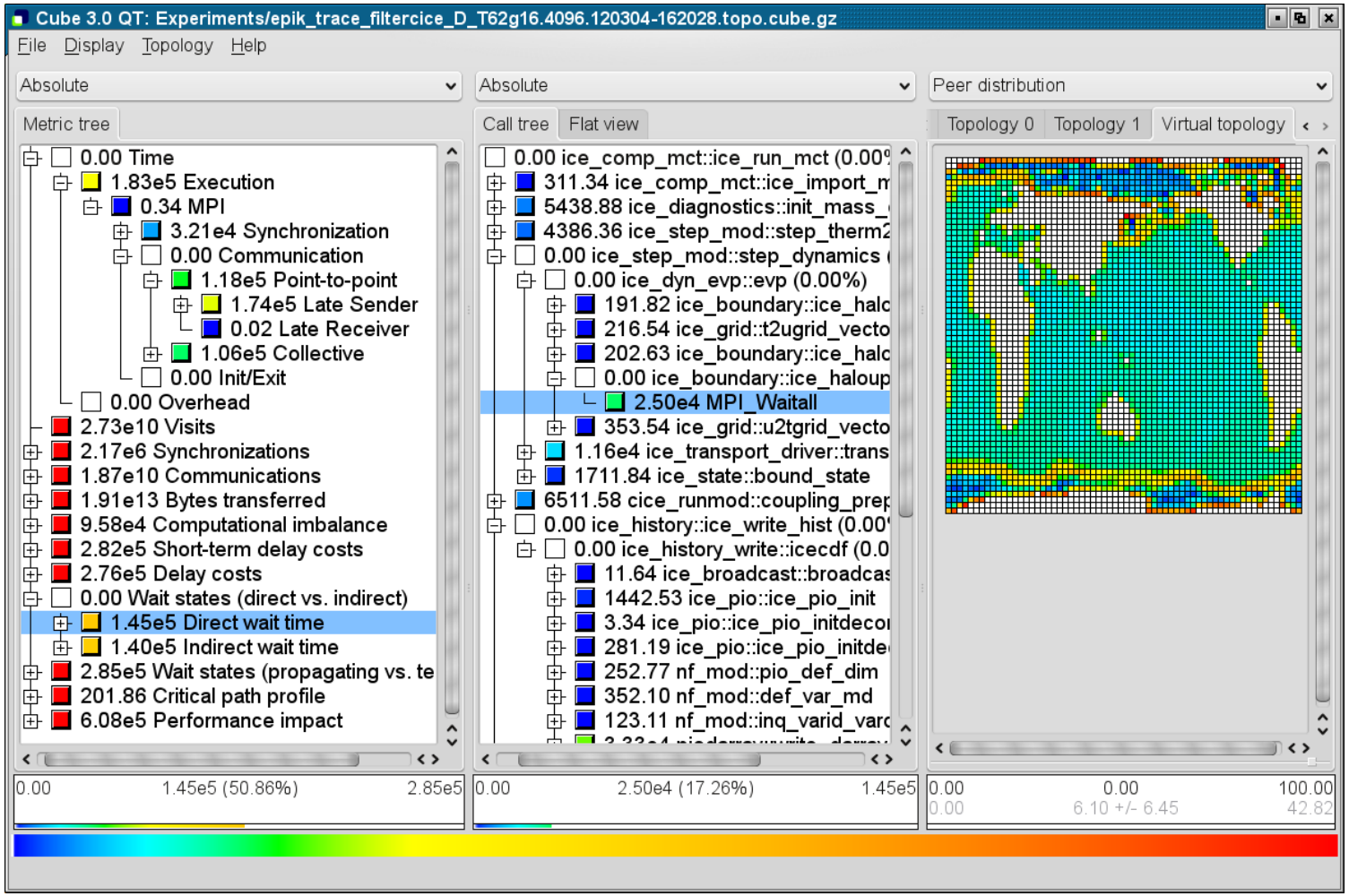
- Classifies wait states into direct and indirect (i.e., caused by other wait states)
- Identifies *delays* (excess computation/communication) as root causes of wait states
- Attributes wait states as *delay costs*

Scalasca Trace Tools features

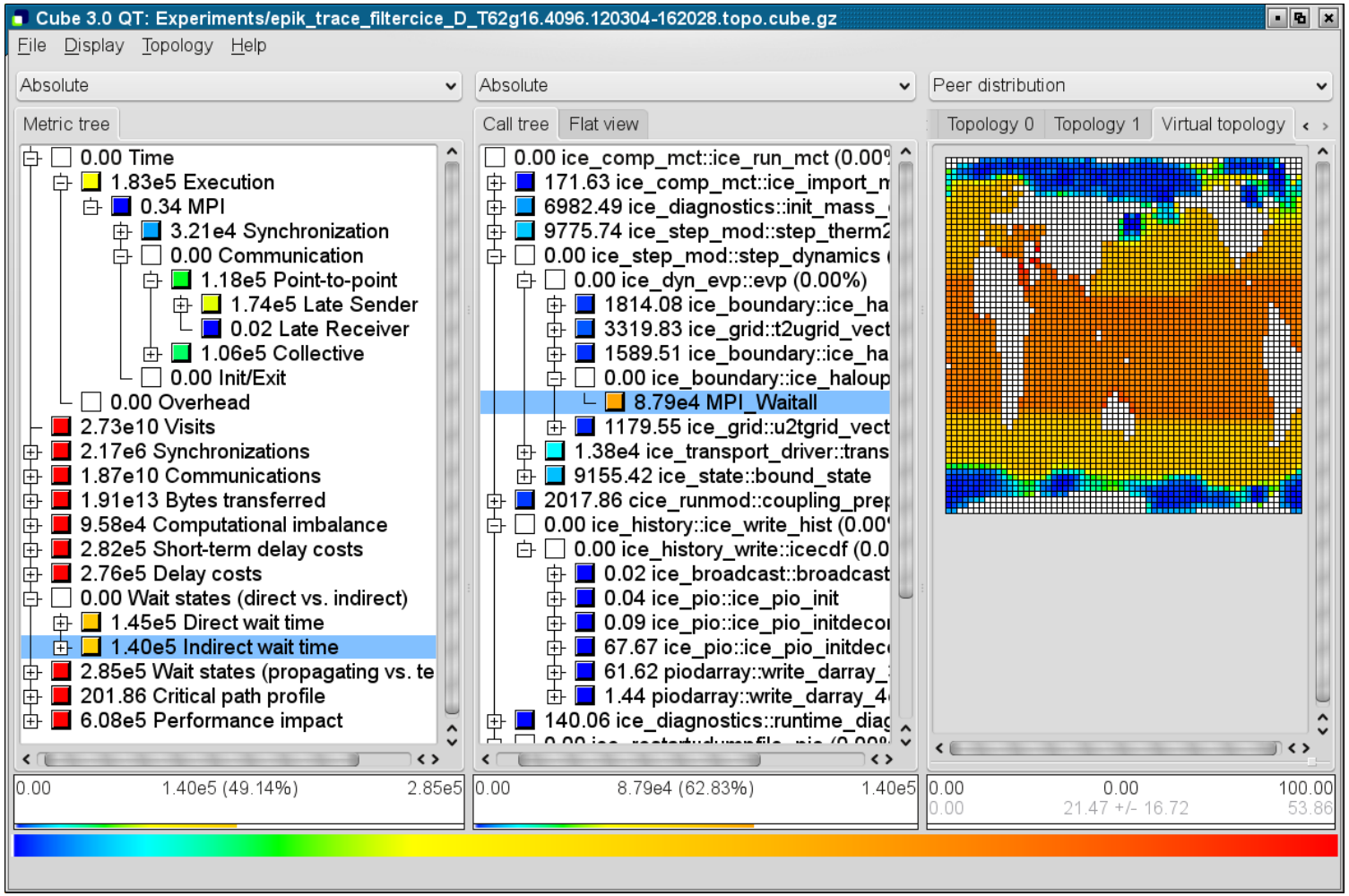
- Open source: 3-clause BSD license
- Portability: supports all major HPC platforms
- Scalability: successful analyses with >1M threads

- Uses Score-P instrumenter & measurement libraries
  - Scalasca v2 core package focuses on trace-based analyses
  - Provides convenience commands for measurement, analysis, and postprocessing
  - Supports common data formats
    - Reads event traces in OTF2 format
    - Writes analysis reports in CUBE4 format

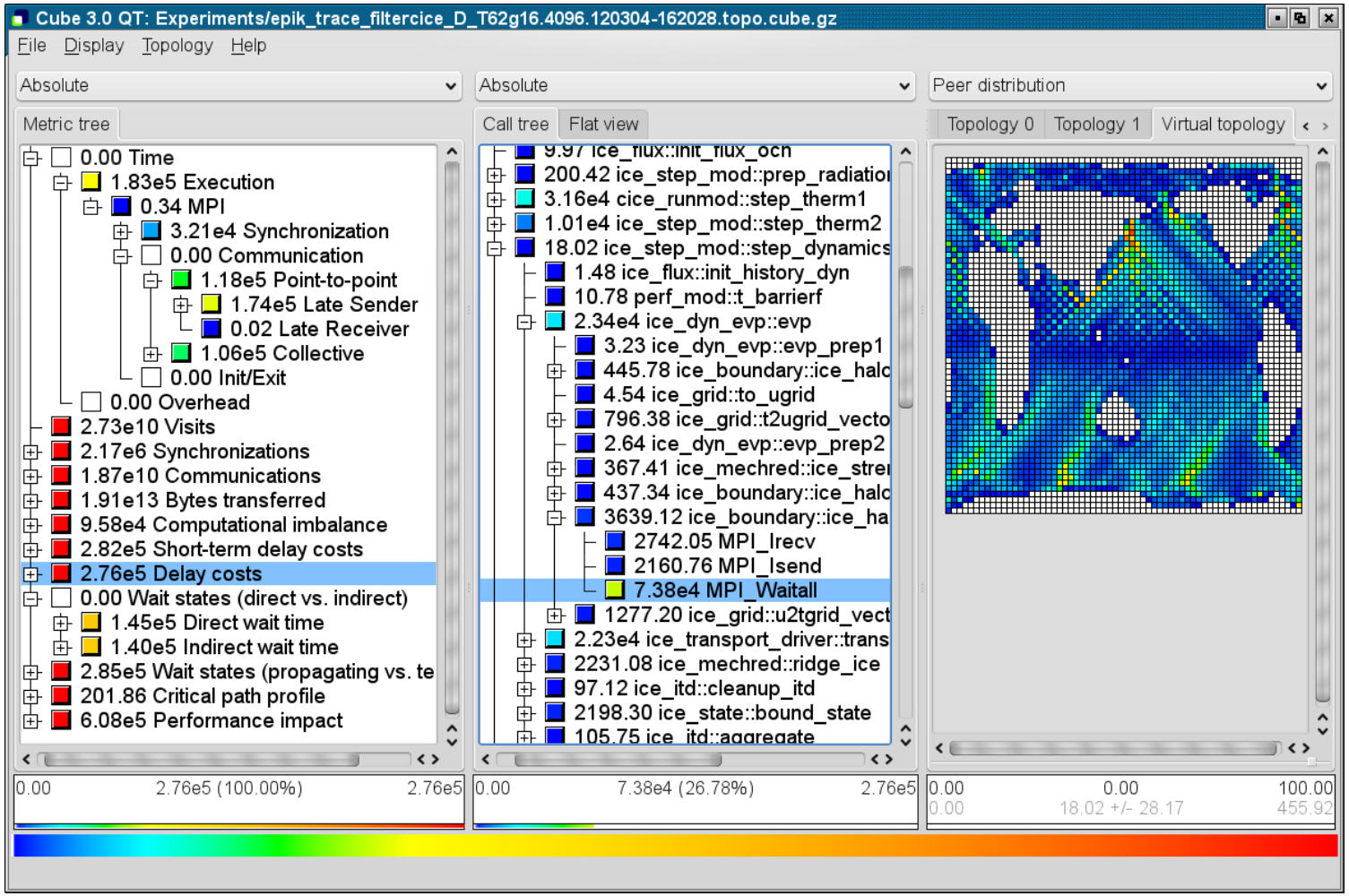
Example: Root-cause analysis - CESM Sea Ice Module



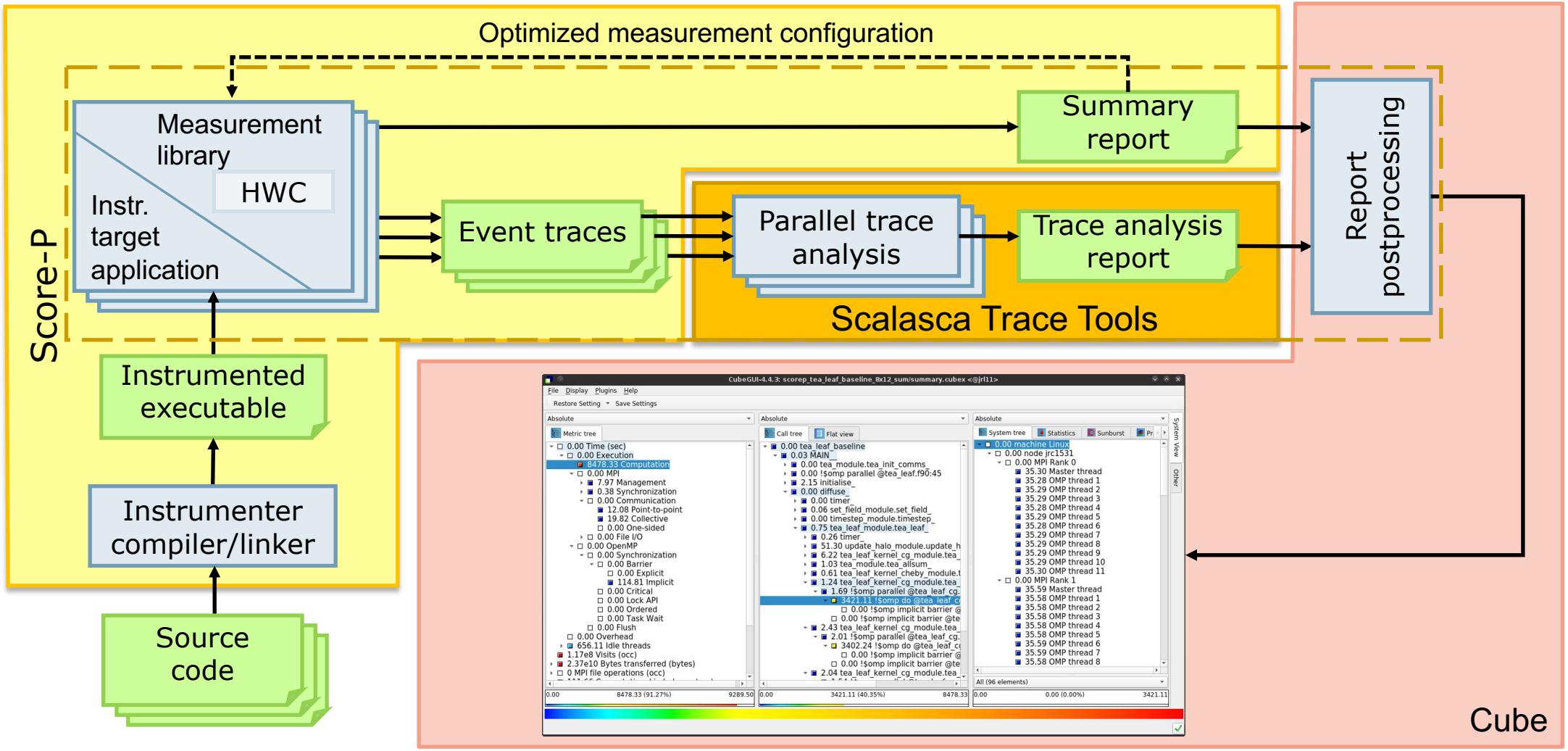
Example: Root-cause analysis - CESM Sea Ice Module



Example: Root-cause analysis - CESM Sea Ice Module

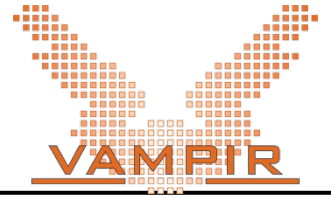


Putting it all together



Vampir, TAU, Extra-P, Darshan

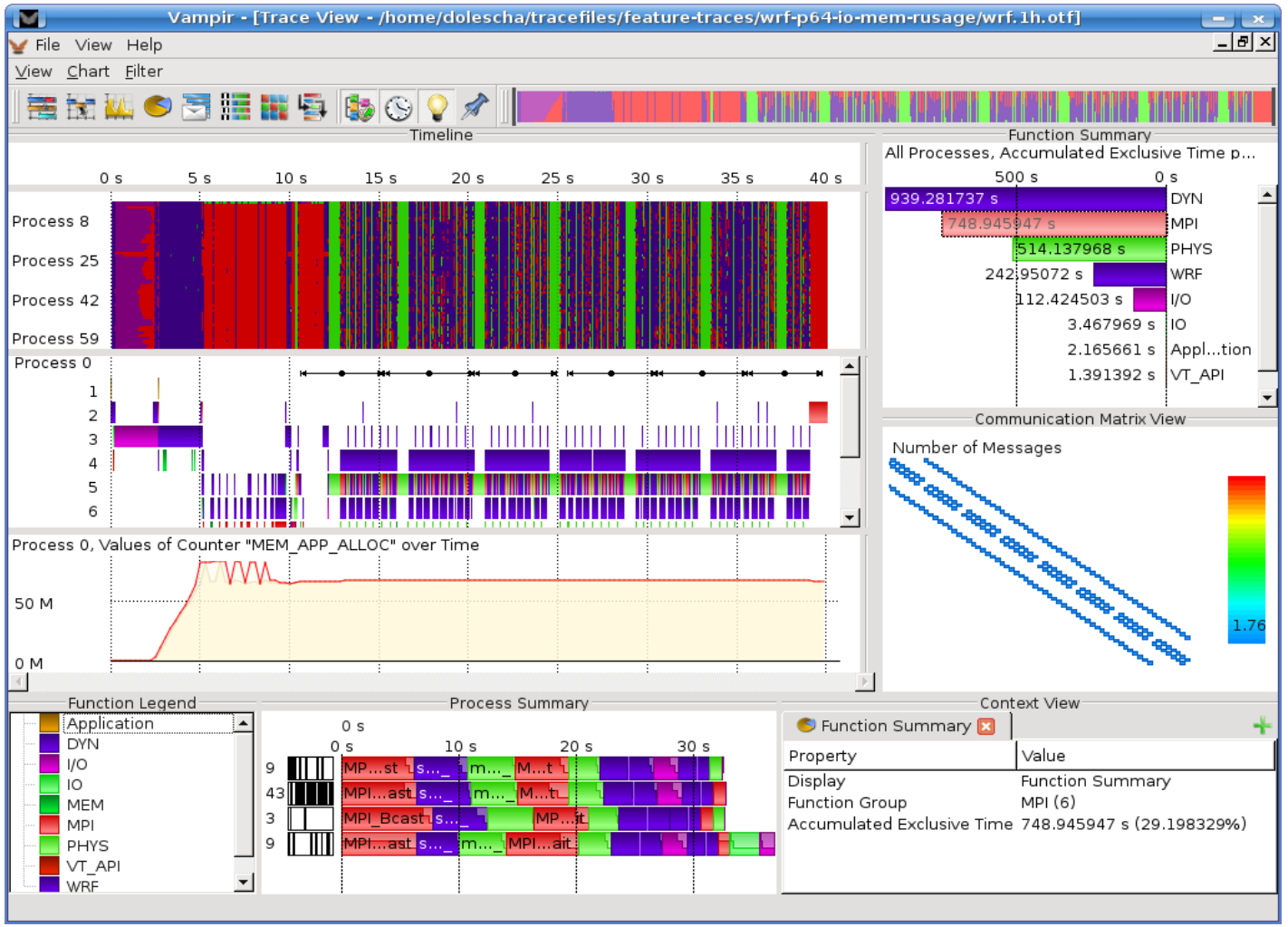
Vampir Event Trace Visualizer



- Offline trace visualization for Score-P's OTF2 trace files
- Visualization of MPI, OpenMP and application events:
  - All diagrams highly customizable (through context menus)
  - Large variety of displays for ANY part of the trace
- http://www.vampir.eu

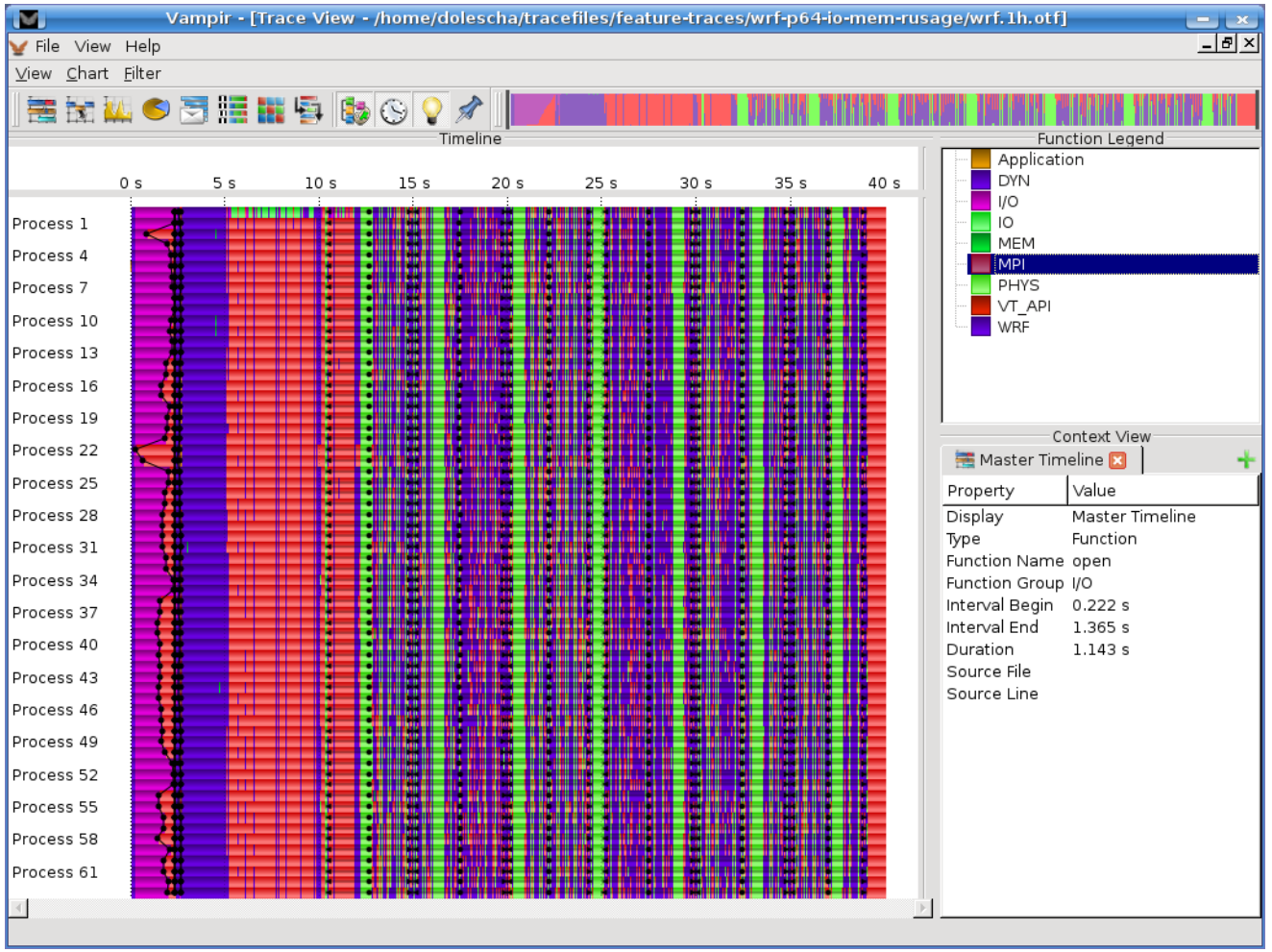
- Advantage:
  - Detailed view of dynamic application behavior
- Disadvantage:
  - Requires event traces (huge amount of data)
  - Completely manual analysis

Vampir Displays



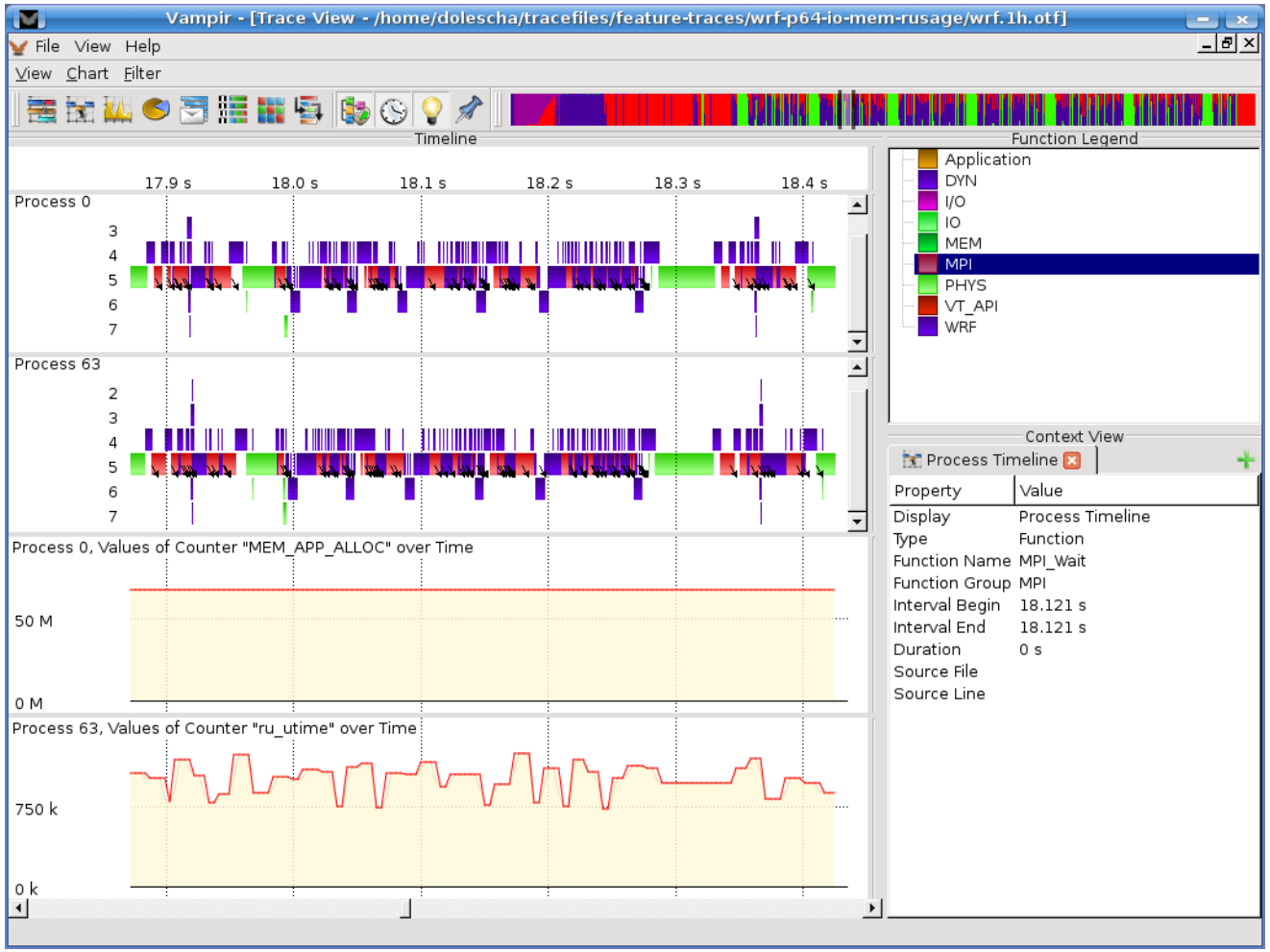
Vampir: Timeline Diagram

- Functions organized into groups
- Coloring by group
- Message lines can be colored by tag or size
- Information about states, messages, collective and I/O operations available through clicking on the representation



Vampir: Process and Counter Timelines

- Process timeline show call stack nesting
- Counter timelines for hardware and software counters

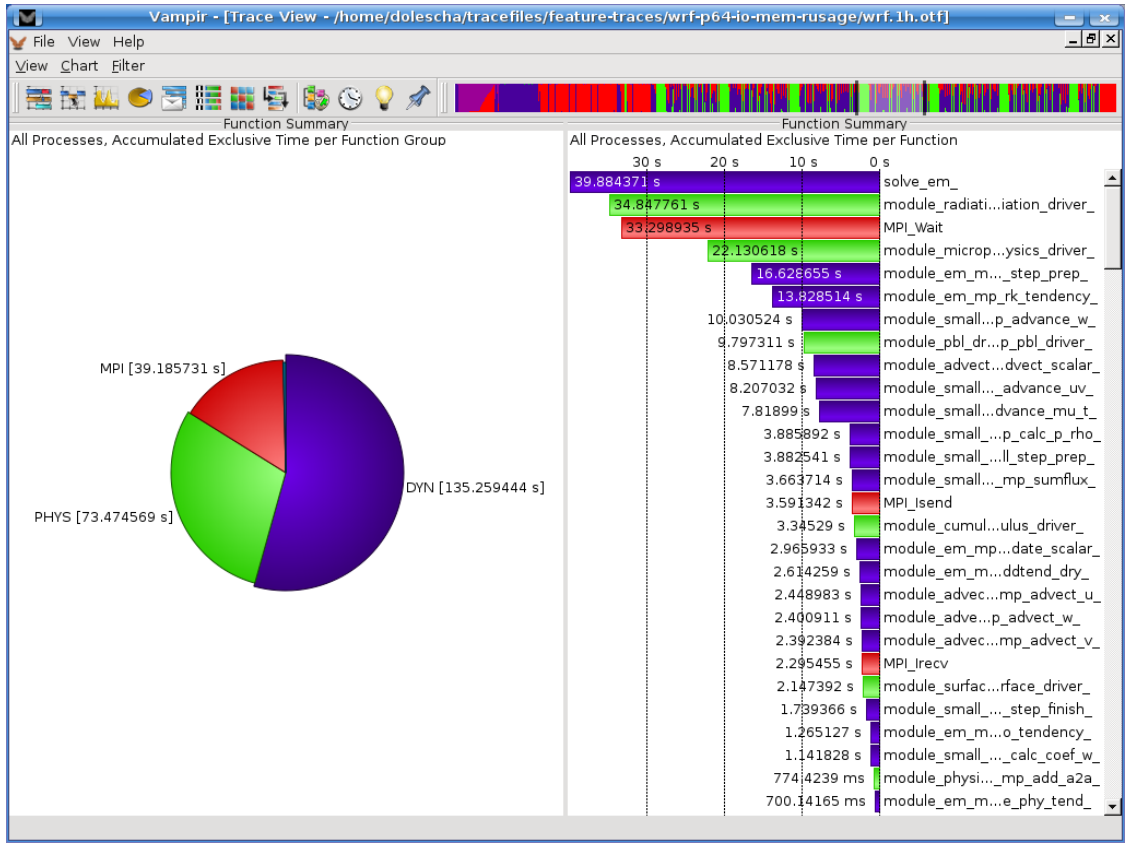


Vampir: Execution Statistics

- Aggregated profiling information:
  execution time,
  number of calls,
  inclusive/exclusive

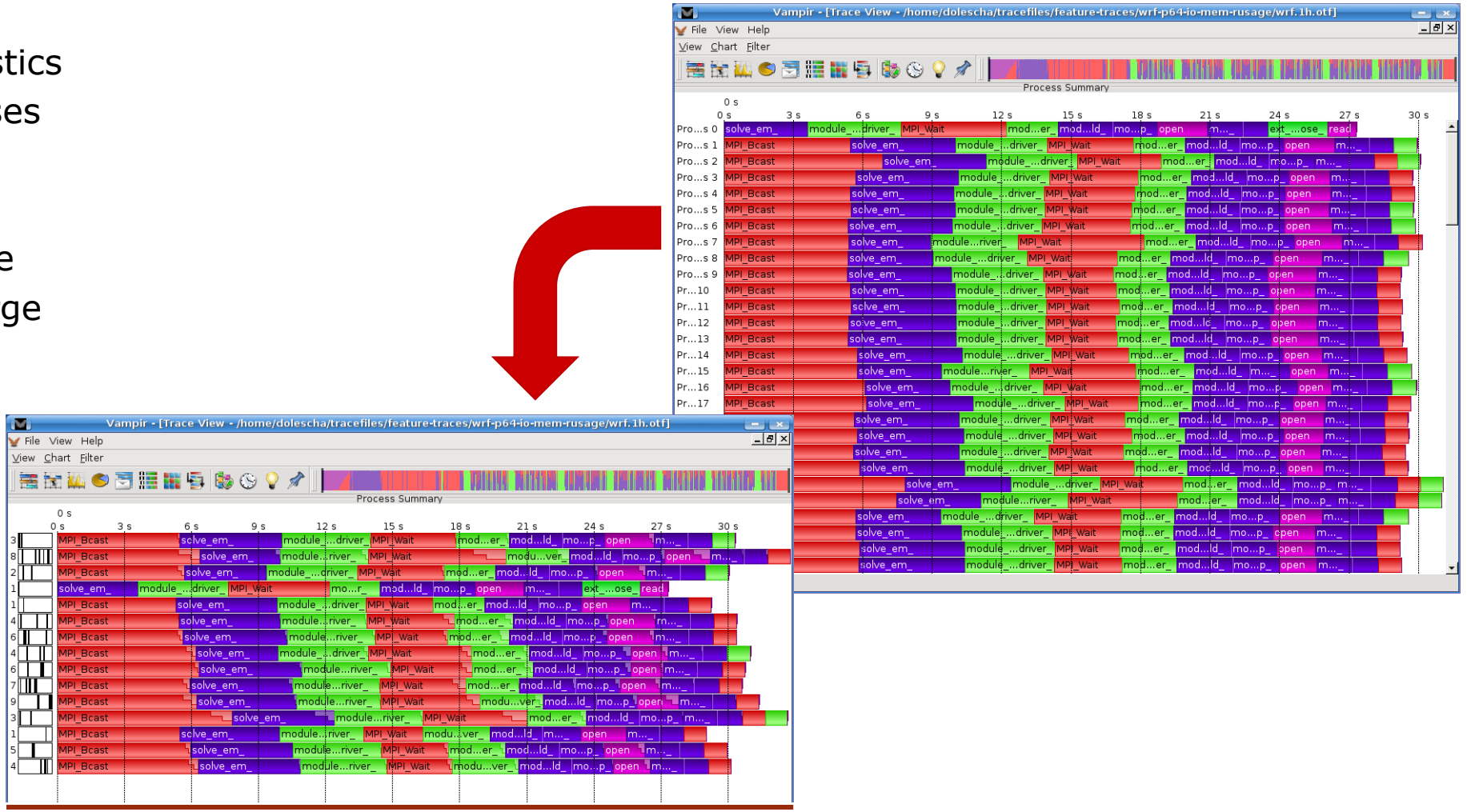
- Available for all / any group (activity) or all routines (symbols)

- Available for any part of the trace
  ⇨ selectable through time line diagram



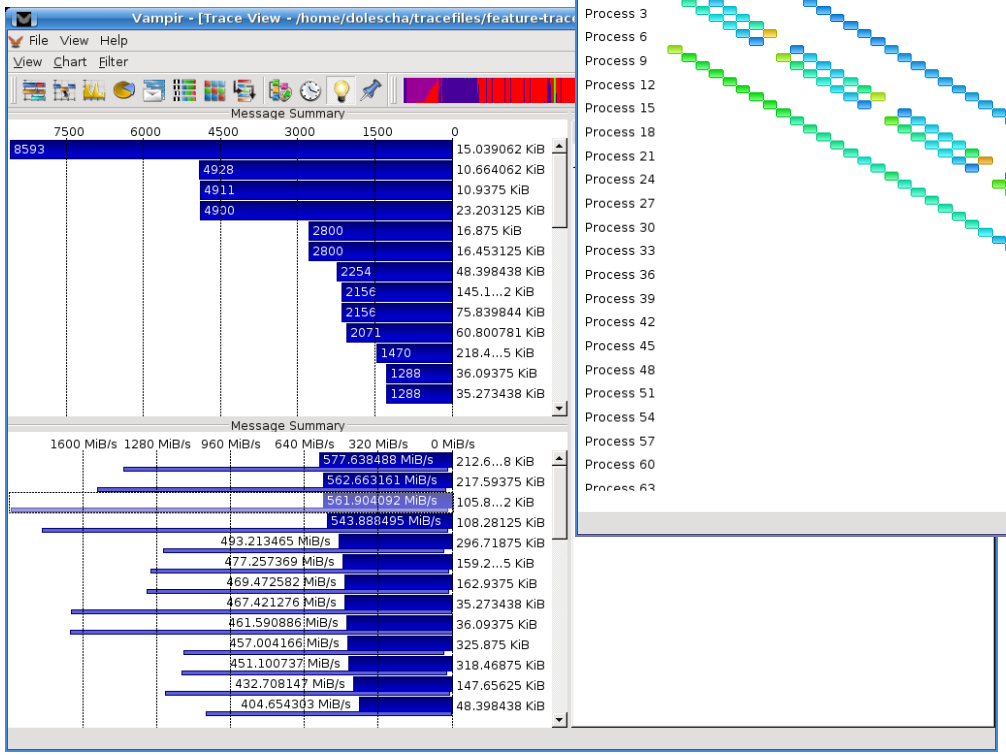
Vampir: Process Summary

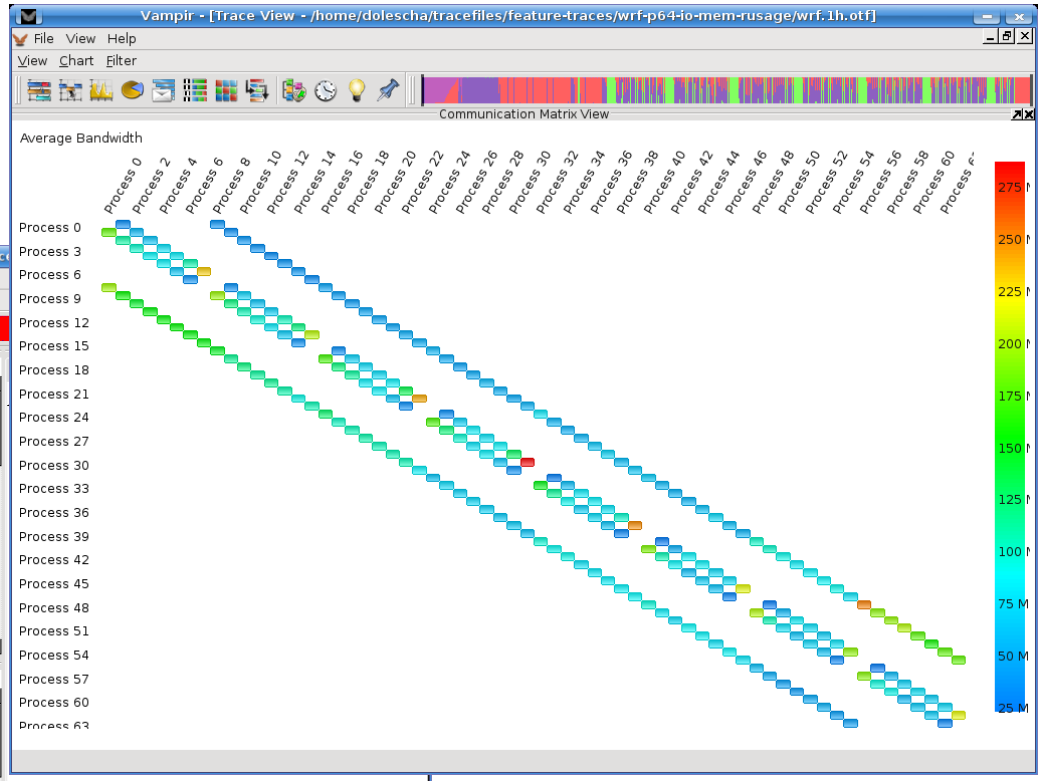
- Execution statistics over all processes for comparison
- Clustering mode available for large process counts



Vampir: Communication Statistics

- Byte and message count, min/max/avg message length and min/max/avg bandwidth for each process pair
- Message length statistics

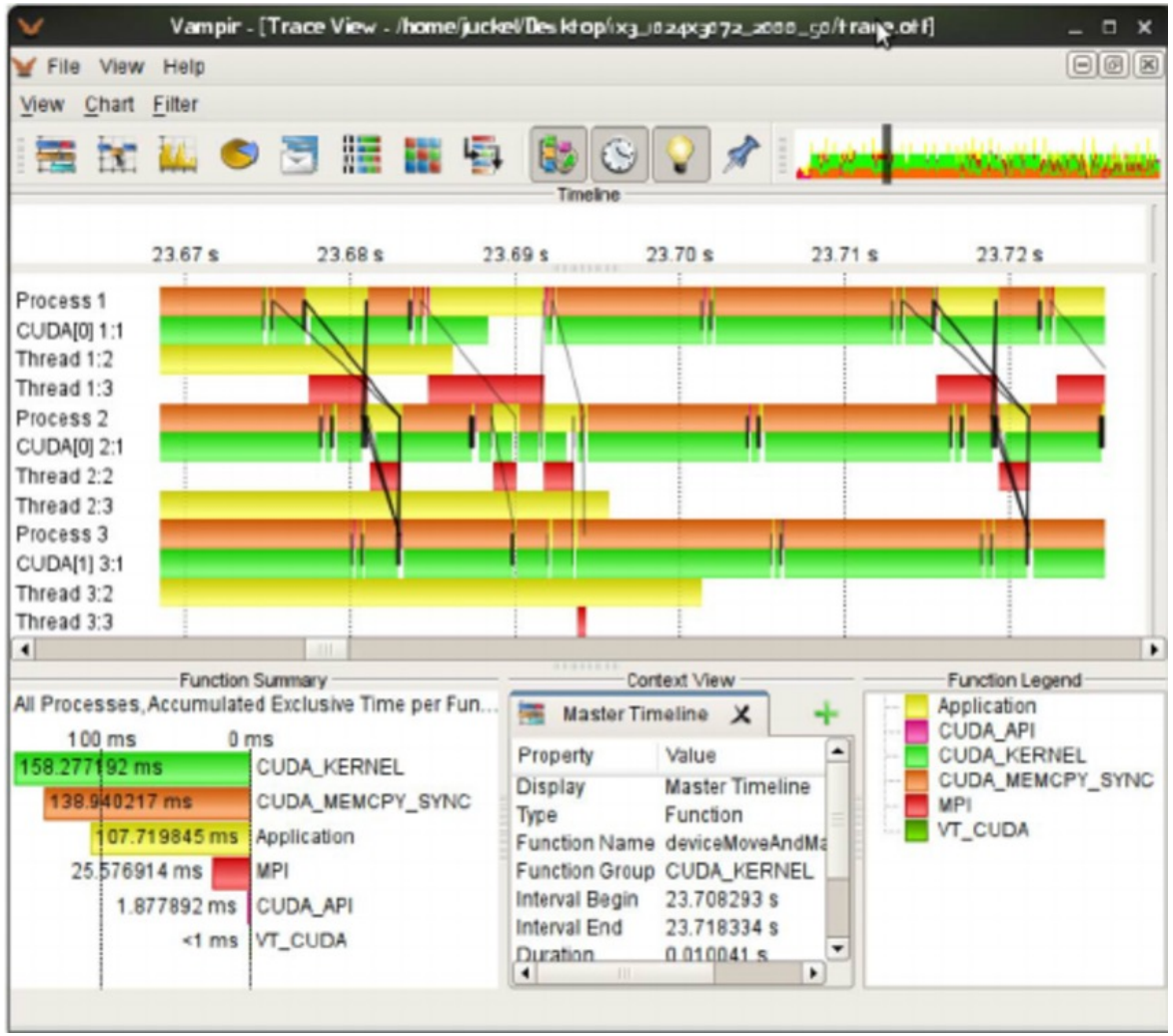




- Available for any part of the trace

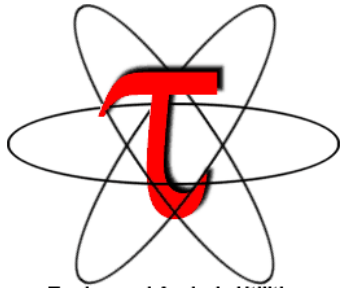
Vampir: CUDA Example

- Detailed information on kernel execution and memory transfers
- All statistics and displays also available for CUDA events



TAU

- Very portable tool set for instrumentation, measurement and analysis of parallel multi-threaded applications
- http://tau.uoregon.edu/

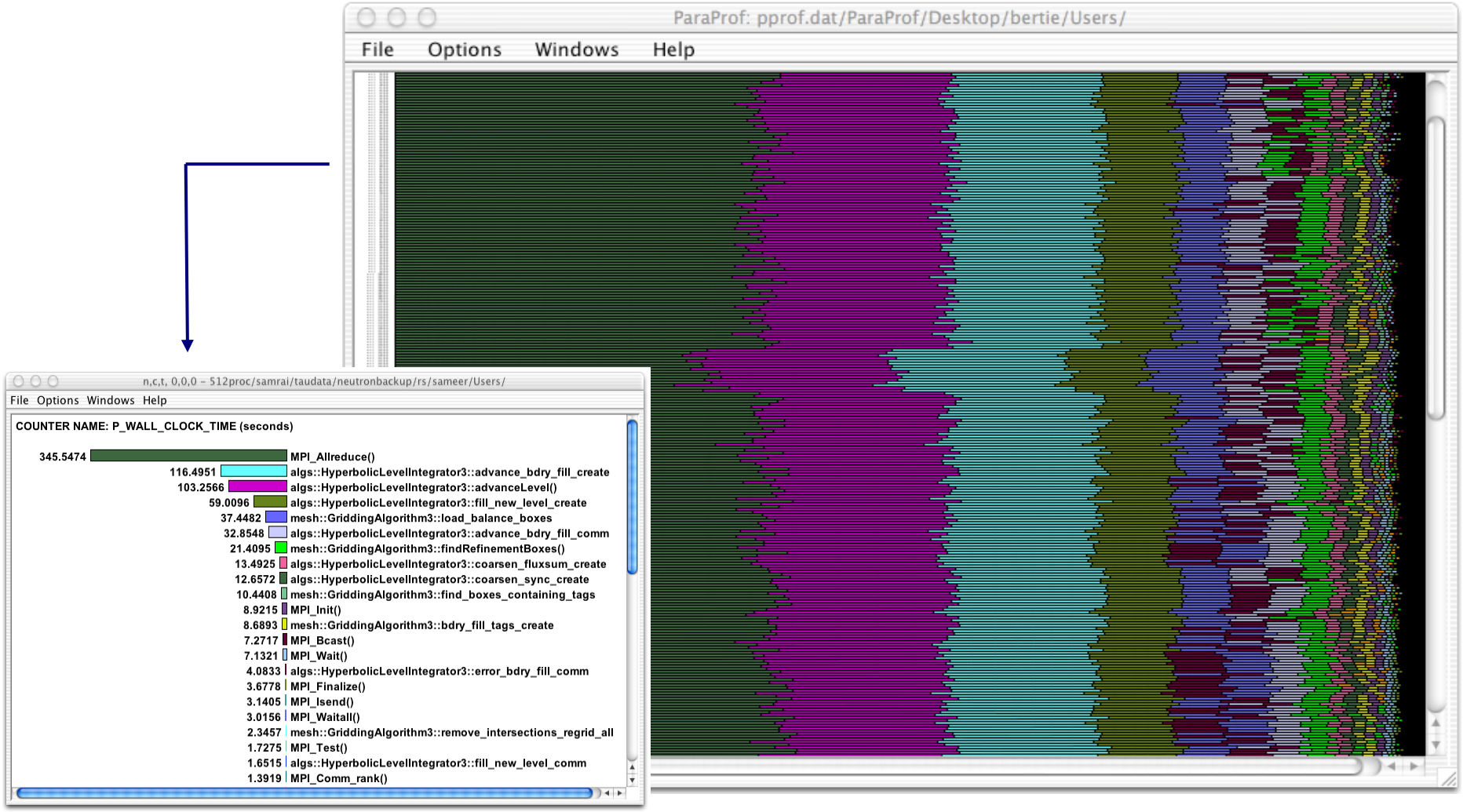


- Supports
  - Various profiling modes and tracing
  - Various forms of code instrumentation
  - C, C++, Fortran, Java, Python
  - MPI, multi-threading (OpenMP, Pthreads, …)
  - Accelerators

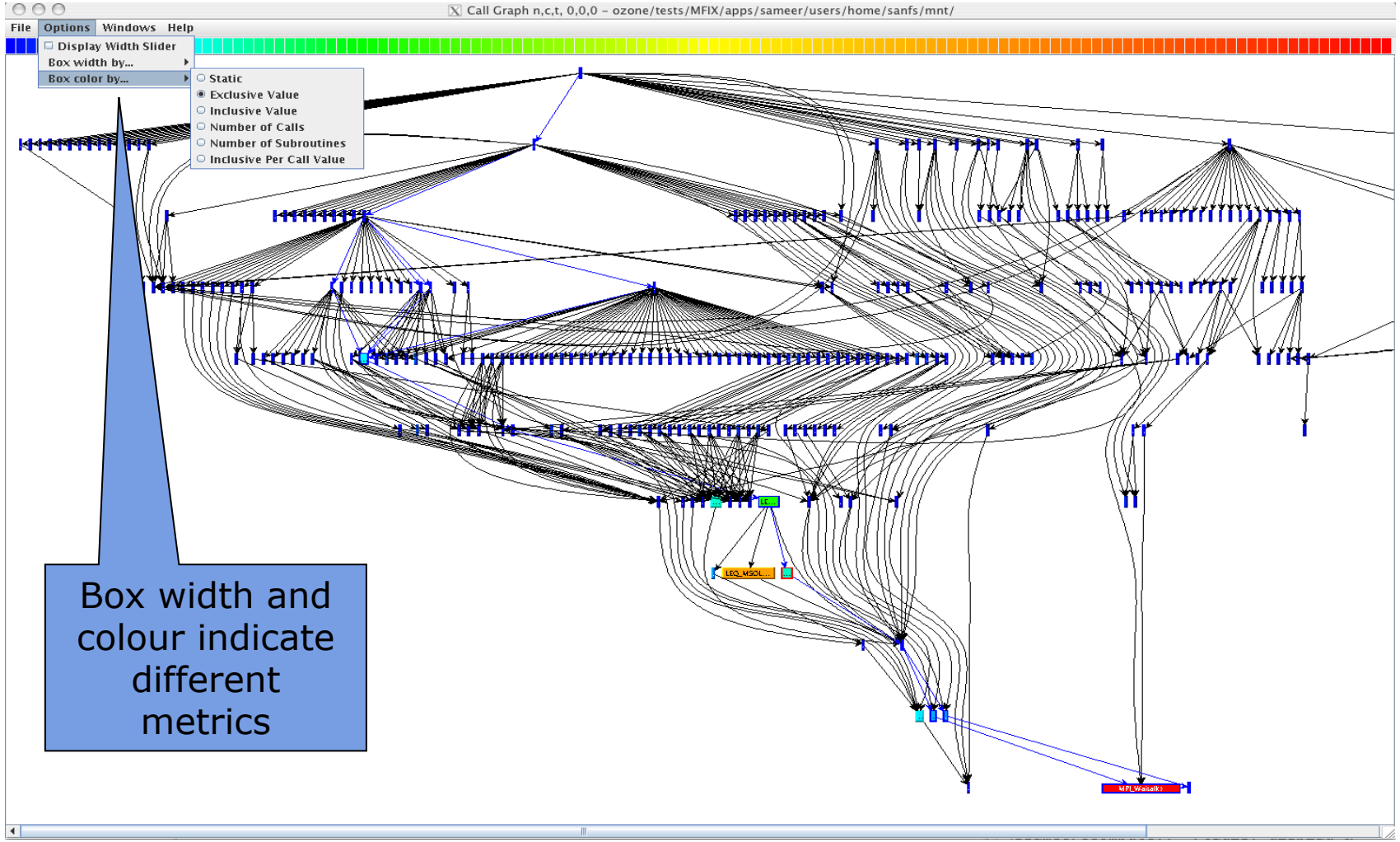
TAU: Instrumentation

- Flexible instrumentation mechanisms at multiple levels
  - Source code
    - manual
    - automatic
      - C, C++, F77/90/95 (Program Database Toolkit (PDT))
      - OpenMP (directive rewriting with Opari)
  - Object code
    - pre-instrumented libraries (e.g., MPI using PMPI)
    - statically-linked and dynamically-loaded (e.g., Python)
  - Executable code
    - dynamic instrumentation (pre-execution) (DynInst)
    - virtual machine instrumentation (e.g., Java using JVMPI)
- Support for performance mapping
- Support for object-oriented and generic programming

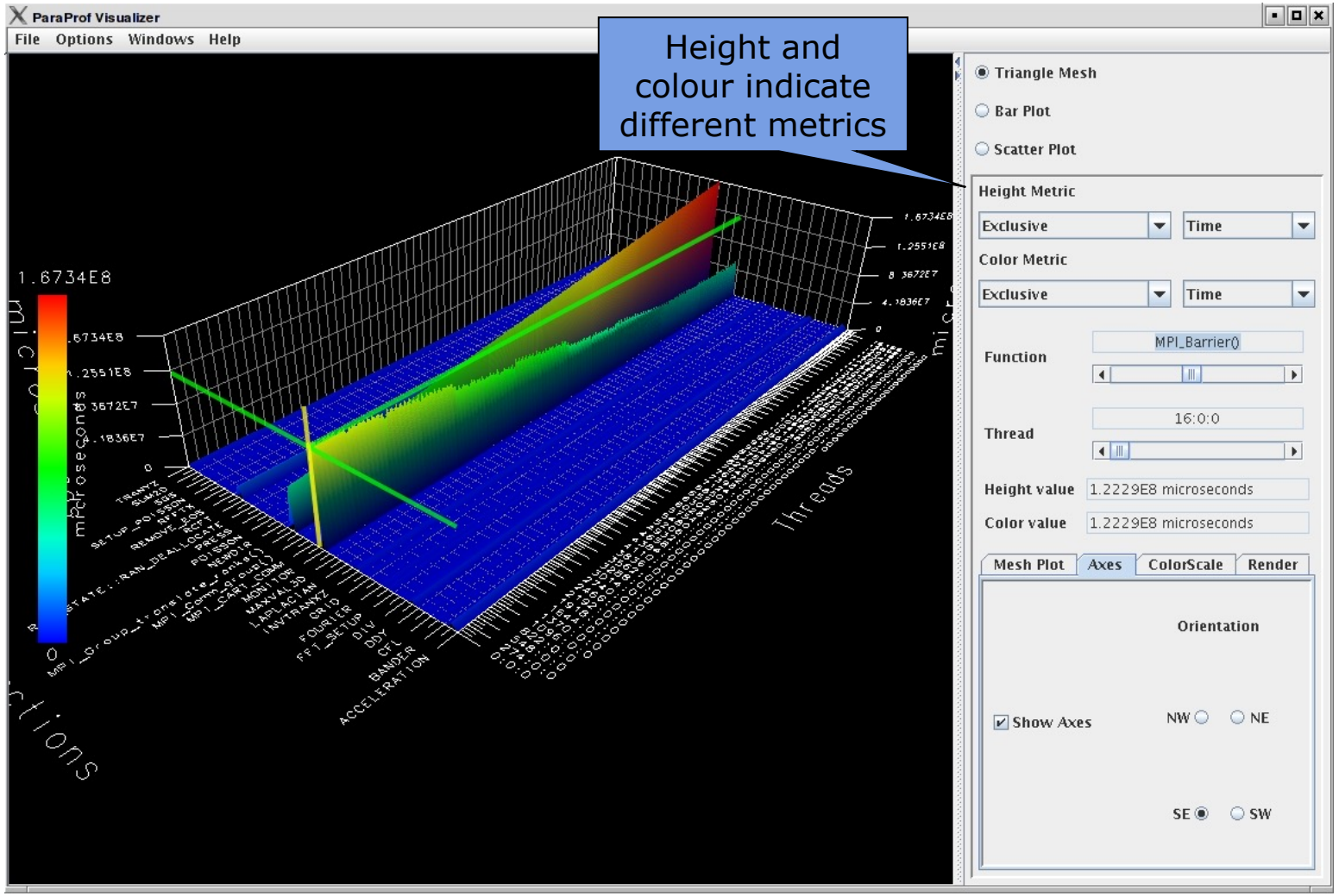
TAU: Basic Profile View



TAU: Callgraph Profile View



TAU: 3D Profile View



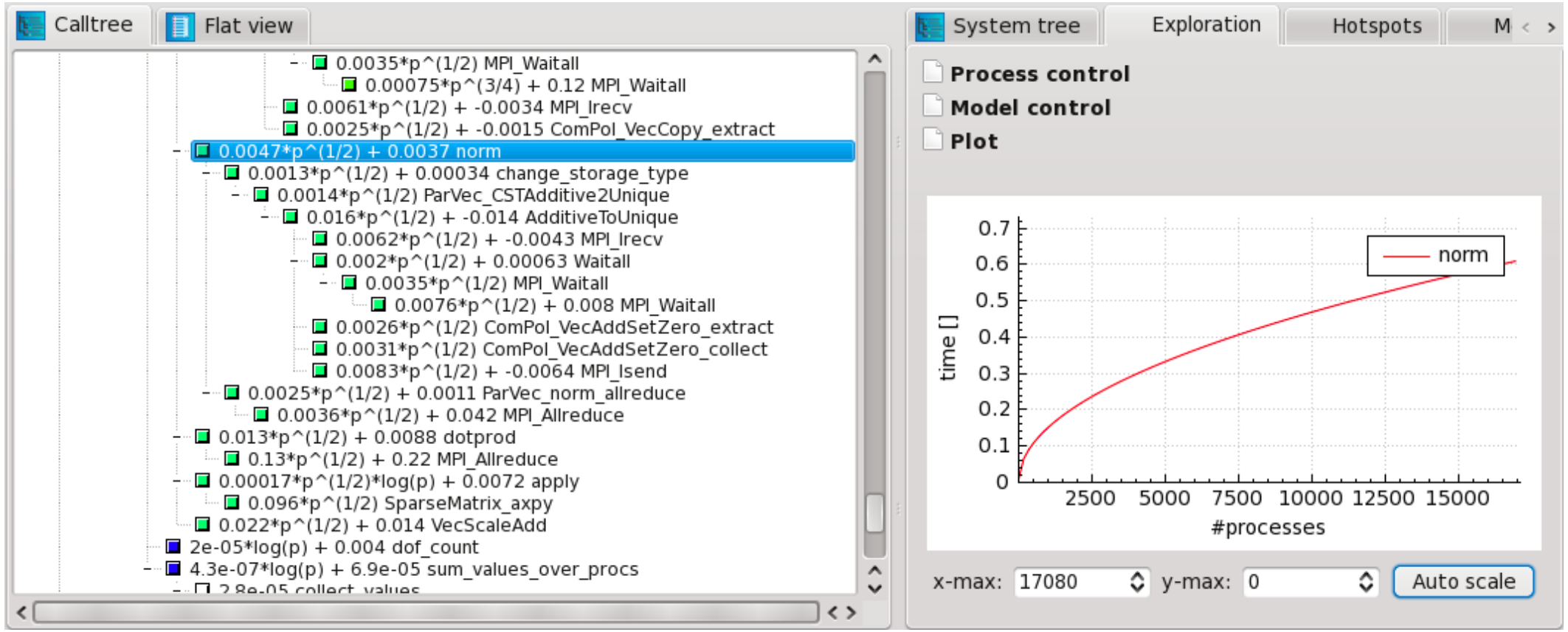
Extra-P

- **Goal**: identification of parts of the program which scaling behavior is unintentionally poor (much worse than expected) by means of automatic performance-modeling

- Supports Linux (x86x86_64/IA64/PPC/Power), Mac OS X (x86_64)
- Accepts input files in the Cube format and processes them into a condensed Cube format containing functions for each metric and call path

- http://www.scalasca.org
- Open Source: BSD 3-Clause License

Interactive exploration of performance models in Extra-P



Darshan

- I/O characterization tool logging parallel application file access
- Summary report provides quick overview of performance issues
- Works on unmodified, optimized executables
- Shows counts of file access operations, times for key operations, histograms of accesses, etc.

- Supports POSIX, MPI-IO, HDF5, PnetCDF, …
- Binary log file written at exit post-processed into PDF report

- http://www.mcs.anl.gov/research/projects/darshan/
- Open Source: installed on many HPC systems

Example Darshan report extract

jobid:	uid:	nprocs: 4096	runtime: 175 seconds