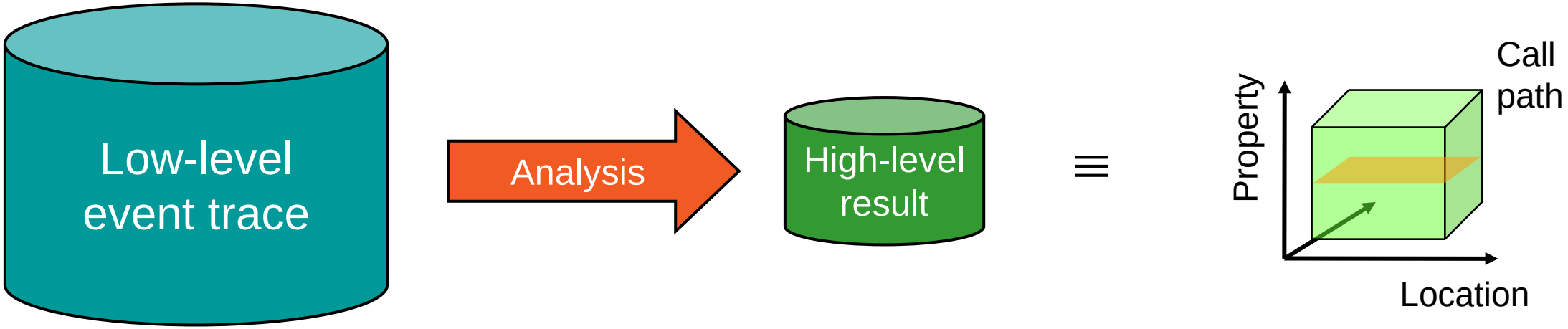
Automatic trace analysis with Scalasca

David Boehme
Lawrence Livermore National Lab

scalasca

Automatic trace analysis

- Idea
  - Automatic search for patterns of inefficient behavior
  - Classification of behavior & quantification of significance



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

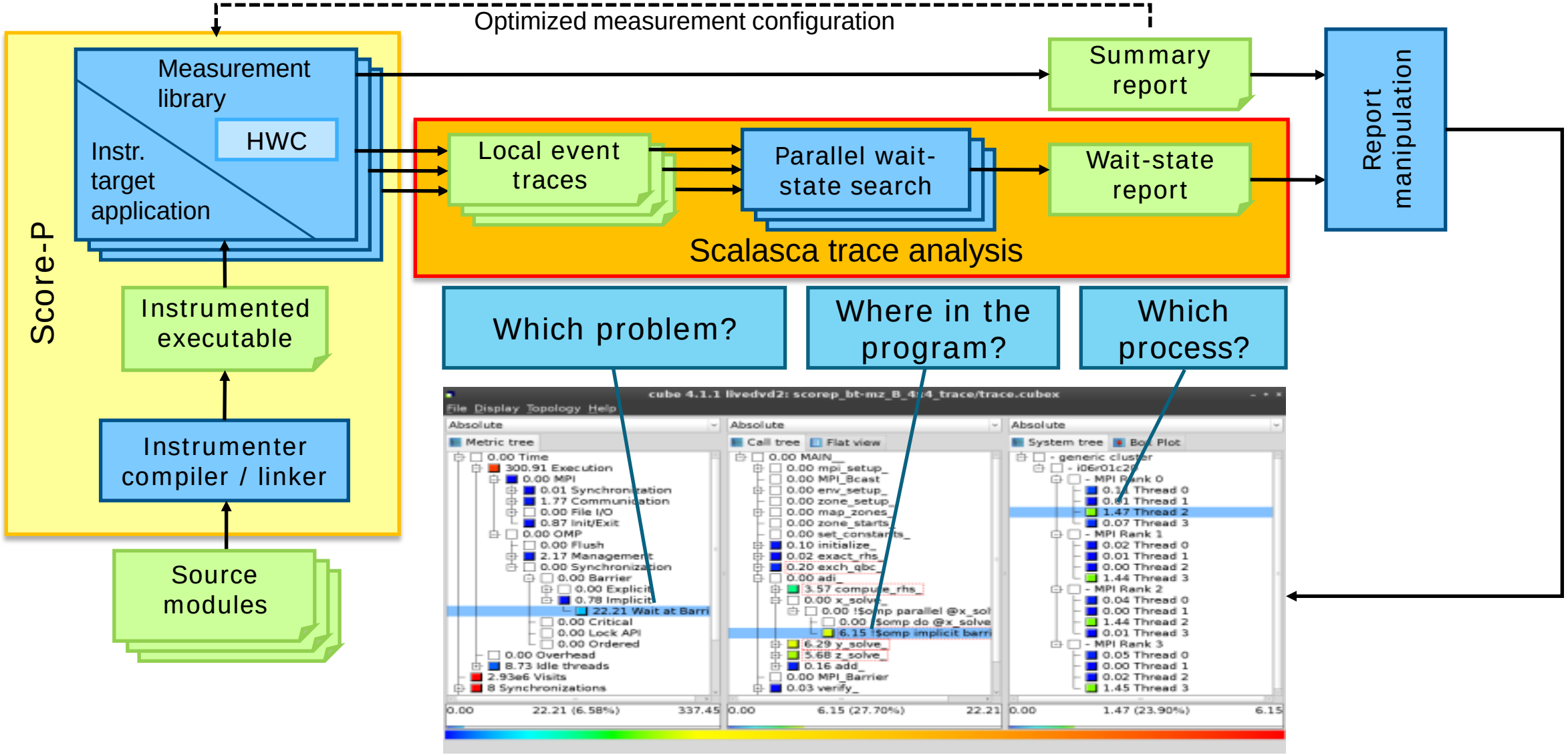
The Scalasca project: Objective

- Development of a **scalable** performance analysis toolset for most popular parallel programming paradigms
- Specifically targeting **large-scale** parallel applications
  - Such as those running on IBM Blue Gene or Cray systems with one million or more processes/threads
- Latest release:
  - Scalasca v2.3.1 coordinated with Score-P v2.0.2 (May 2016)

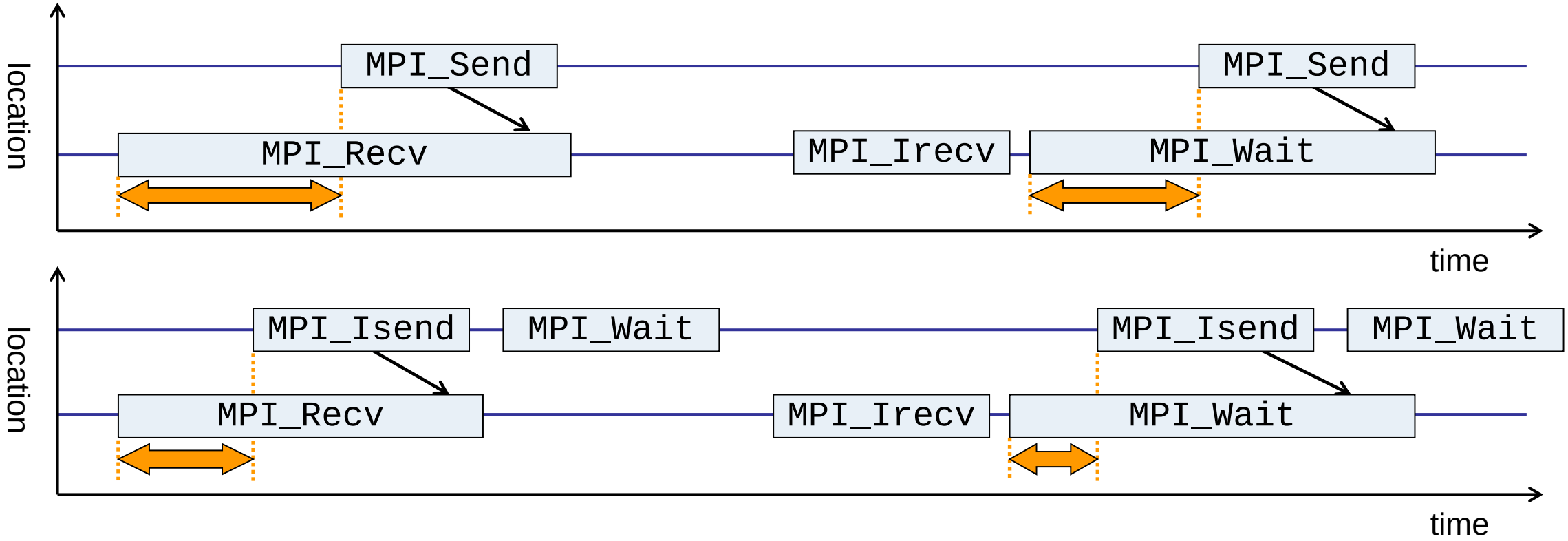
Scalasca features

- Open source, 3-clause BSD license
- Fairly portable
  - IBM Blue Gene, Cray XT/XE/XK/XC, SGI Altix, Fujitsu FX10/100 & K computer, Linux clusters (x86, Power, ARM), Intel Xeon Phi, ...
- Uses Score-P instrumenter & measurement libraries
  - Scalasca 2 core package focuses on trace-based analyses
  - Supports common data formats
    - Reads event traces in OTF2 format
    - Writes analysis reports in CUBE4 format
- Current limitations:
  - Unable to handle traces containing CUDA or SHMEM events, or OpenMP nested parallelism
  - PAPI/rusage metrics for trace events are ignored

Scalasca workflow

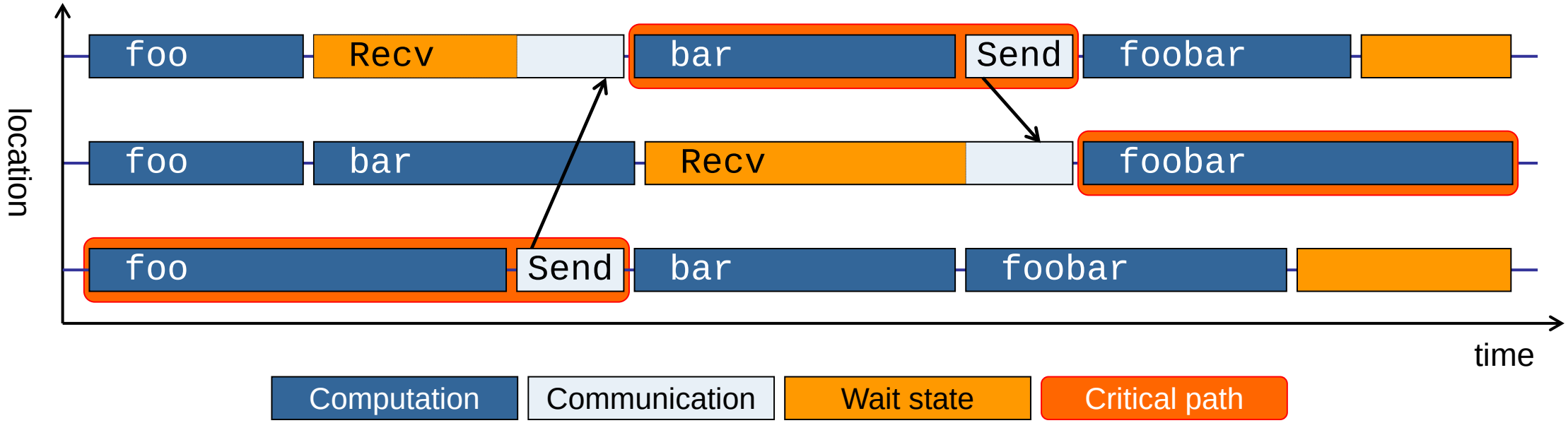


Example: "*Late Sender*" wait state



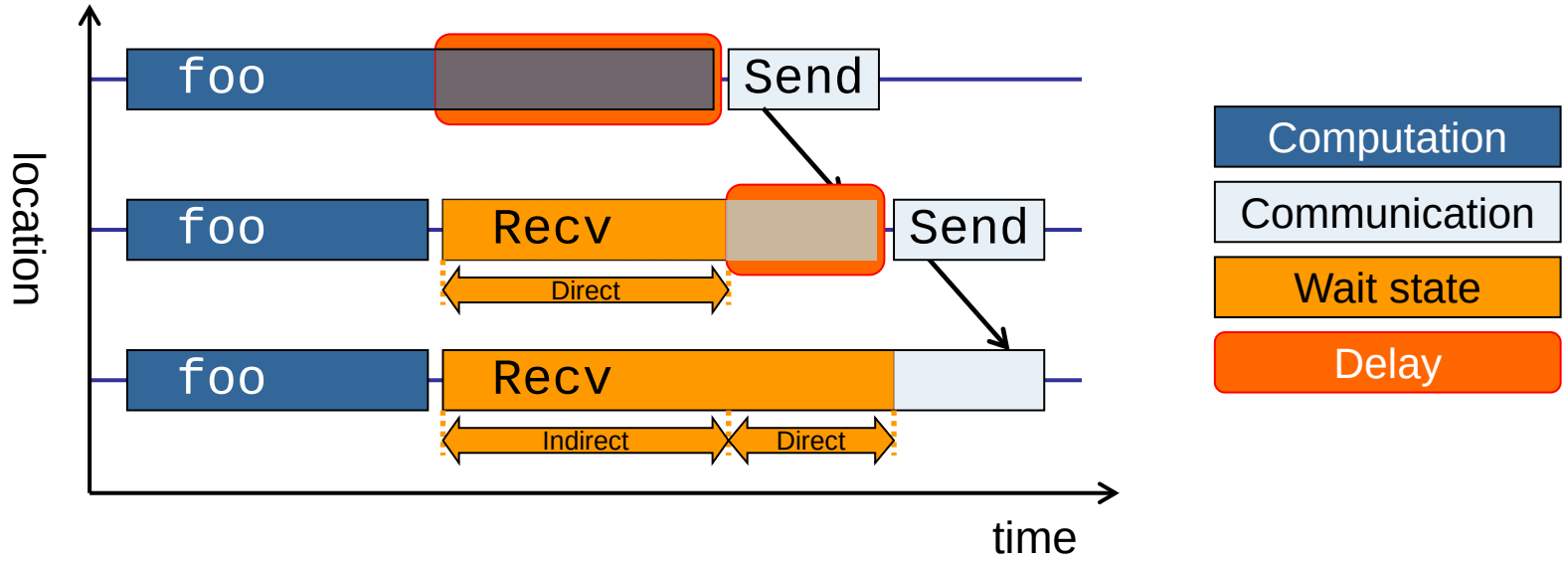
- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication

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*

Hands-on:
NPB-MZ-MPI / BT

Scalasca command – One command for (almost) everything

```
% scalasca
Scalasca 2.3.1
Toolset for scalable performance analysis of large-scale parallel applications
usage: scalasca [OPTION]... ACTION <argument>...
    1. prepare application objects and executable for measurement:
       scalasca -instrument <compile-or-link-command> # skin (using scorep)
    2. run application under control of measurement system:
       scalasca -analyze <application-launch-command> # scan
    3. interactively explore measurement analysis report:
       scalasca -examine <experiment-archive|report>  # square

Options:
   -c, --show-config     show configuration summary and exit
   -h, --help            show this help and exit
   -n, --dry-run         show actions without taking them
       --quickref        show quick reference guide and exit
       --remap-specfile  show path to remapper specification file and exit
   -v, --verbose         enable verbose commentary
   -V, --version         show version information and exit
```

- The ‘`scalasca -instrument`’ command is deprecated and only provided for backwards compatibility with Scalasca 1.x., recommended: use Score-P instrumenter directly

Scalasca compatibility command: skin

```
% skin
Scalasca 2.3.1: application instrumenter (using Score-P instrumenter)
usage: skin [-v] [-comp] [-pdt] [-pomp] [-user] [--*] <compile-or-link-command>
  -comp={all|none|...}: routines to be instrumented by compiler [default: all]
                        (... custom instrumentation specification depends on compiler)
  -pdt:  process source files with PDT/TAU instrumenter
  -pomp: process source files for POMP directives
  -user: enable EPIK user instrumentation API macros in source code
  -v:    enable verbose commentary when instrumenting

  --*:   options to pass to Score-P instrumenter
```

- Scalasca application instrumenter
  - Provides compatibility with Scalasca 1.x
  - **Deprecated! Use Score-P instrumenter directly.**

Scalasca convenience command: scan

```
% scan
Scalasca 2.3.1: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
       where {options} may include:
  -h    Help: show this brief usage message and exit.
  -v    Verbose: increase verbosity.
  -n    Preview: show command(s) to be launched but don't execute.
  -q    Quiescent: execution with neither summarization nor tracing.
  -s    Summary: enable runtime summarization. [Default]
  -t    Tracing: enable trace collection and analysis.
  -a    Analyze: skip measurement to (re-)analyze an existing trace.
  -e exptdir   : Experiment archive to generate and/or analyze.
                 (overrides default experiment archive title)
  -f filtfile  : File specifying measurement filter.
  -l lockfile  : File that blocks start of measurement.
  -m metrics   : Metric specification for measurement.
```

- Scalasca measurement collection & analysis nexus

Scalasca advanced command: scout - Scalasca automatic trace analyzer

```
% scout.hyb --help
SCOUT   Copyright (c) 1998-2016 Forschungszentrum Juelich GmbH
        Copyright (c) 2009-2014 German Research School for Simulation
                                Sciences GmbH

Usage: <launchcmd> scout.hyb [OPTION]... <ANCHORFILE | EPIK DIRECTORY>
Options:
  --statistics       Enables instance tracking and statistics [default]
  --no-statistics    Disables instance tracking and statistics
  --critical-path    Enables critical-path analysis [default]
  --no-critical-path Disables critical-path analysis
  --rootcause        Enables root-cause analysis [default]
  --no-rootcause     Disables root-cause analysis
  --single-pass      Single-pass forward analysis only
  --time-correct     Enables enhanced timestamp correction
  --no-time-correct  Disables enhanced timestamp correction [default]
  --verbose, -v      Increase verbosity
  --help             Display this information and exit
```

- Provided in serial (.ser), OpenMP (.omp), MPI (.mpi) and MPI+OpenMP (.hyb) variants

Scalasca advanced command: clc_synchronize

- Scalasca trace event timestamp consistency correction

```
Usage: <launchcmd> clc_synchronize.hyb <ANCHORFILE | EPIK_DIRECTORY>
```

  - Provided in MPI (.mpi) and MPI+OpenMP (.hyb) variants
  - Takes as input a trace experiment archive where the events may have timestamp inconsistencies
    - E.g., multi-node measurements on systems without adequately synchronized clocks on each compute node
  - Generates a new experiment archive (always called ./clc_sync) containing a trace with event timestamp inconsistencies resolved
    - E.g., suitable for detailed examination with a time-line visualizer

Scalasca convenience command: square

```
% square
Scalasca 2.3.1: analysis report explorer
usage: square [-v] [-s] [-f filtfile] [-F] <experiment archive | cube file>
   -c <none | quick | full> : Level of sanity checks for newly created reports
   -F                       : Force remapping of already existing reports
   -f filtfile              : Use specified filter file when doing scoring
   -s                       : Skip display and output textual score report
   -v                       : Enable verbose mode
   -n                       : Do not include idle thread metric
```

- Scalasca analysis report explorer

Automatic measurement configuration

- scan configures Score-P measurement by automatically setting some environment variables and exporting them
  - E.g., experiment title, profiling/tracing mode, filter file, …
  - Precedence order:
    - Command-line arguments
    - Environment variables already set
    - Automatically determined values
- Also, scan includes consistency checks and prevents corrupting existing experiment directories
- For tracing experiments, after trace collection completes then automatic parallel trace analysis is initiated
  - Uses identical launch configuration to that used for measurement (i.e., the same allocated compute resources)

Setup environment

- Remember load modules to add local tool installations to $PATH
  - Must match the compiler you intend to use (Intel or GCC)

```
% use scorep-2.0.2-mvapich2-gnu-2.2
% use scalasca-2.3.1-mvapich2-gnu-2.2
```

- Change to directory containing NPB3.3-MZ-MPI sources
- Existing instrumented executable in bin.scorep/ directory can be reused

```
% cd $HOME/scratch/NPB3.3-MZ-MPI
```

Performance analysis steps

- 0.0 Reference preparation for validation

- 1.0 Program instrumentation
- 1.1 Summary measurement collection
- 1.2 Summary analysis report examination

- 2.0 Summary experiment scoring
- 2.1 Summary measurement collection with filtering
- 2.2 Filtered summary analysis report examination

- 3.0 Event trace collection
- 3.1 Event trace examination & analysis

BT-MZ trace measurement collection...

```
% cd bin.scorep
% cp ../jobscript/darwin/scalasca.sbatch .
% vi scalasca.sbatch

 [...]

export SCOREP_FILTERING_FILE=../config/scorep.filt
export SCOREP_TOTAL_MEMORY=21MB

# Scalasca configuration
export SCAN_ANALYZE_OPTS="--time-correct"

scalasca –analyze –t srun –n8 –ppdebug./bt-mz_${CLASS}.${PROCS}
```

- Change to directory with the executable and edit the job script

```
% sbatch scalasca.sbatch
```

- Submit the job

BT-MZ trace measurement ... collection

```
S=C=A=N: Scalasca 2.3.1 trace collection and analysis
S=C=A=N: ./scorep_bt-mz_B_8x4_trace experiment archive
S=C=A=N: Sat Jul  2 16:20:56 2016: Collect start \
  mpirun -np 8 ./bt-mz_B.8


 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP \
 Benchmark

 Number of zones:   8 x   8
 Iterations: 200    dt:   0.000300
 Number of active processes:     8

 [... More application output ...]

S=C=A=N: Sat Jul  2 16:21:11 2016: Collect done (status=0) 15s
```

- Starts measurement with collection of trace files ...

BT-MZ trace measurement ... analysis

```
S=C=A=N: Sat Jul  2 16:21:11 2016: Analyze start \
 mpirun -np 8 scout.hyb ./scorep_bt-mz_B_8x4_trace/traces.otf2
SCOUT   Copyright (c) 1998-2016 Forschungszentrum Juelich GmbH
        Copyright (c) 2009-2014 German Research School for Simulation
                                Sciences GmbH

Analyzing experiment archive ./scorep_bt-mz_B_8x4_trace/traces.otf2

Opening experiment archive ... done (0.073s).
Reading definition data    ... done (0.053s).
Reading event trace data   ... done (0.106s).
Preprocessing              ... done (0.125s).
Analyzing trace data       ... done (2.778s).
Writing analysis report    ... done (0.260s).

Max. memory usage         : 171.227MB

Total processing time     : 3.436s
S=C=A=N: Sat Jul  2 16:21:16 2016: Analyze done (status=0) 5s
```

- Continues with automatic (parallel) analysis of trace files

BT-MZ trace analysis report exploration

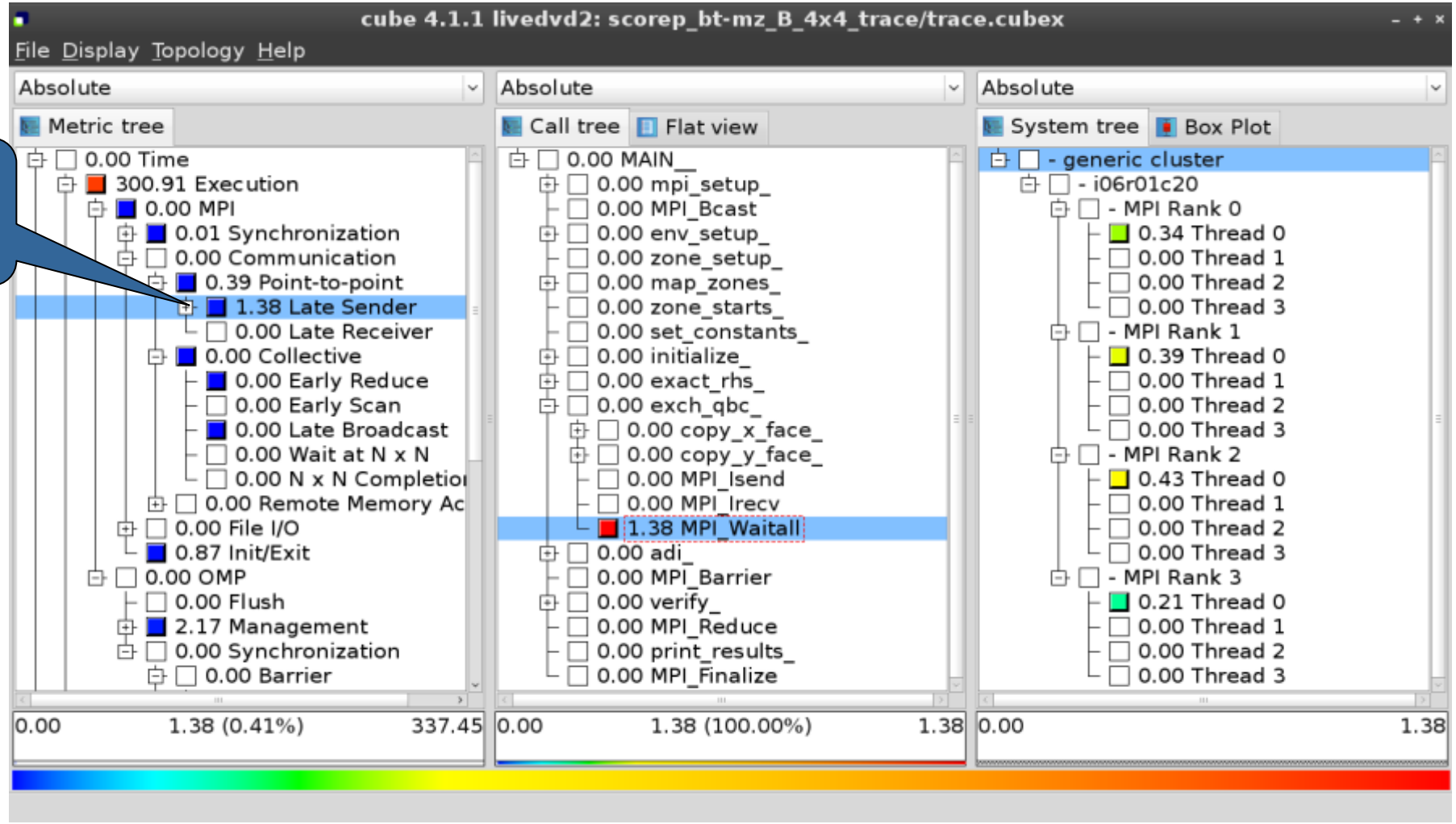
- Produces trace analysis report in the experiment directory containing trace-based wait-state metrics

```
% square  scorep_bt-mz_B_8x4_trace
INFO: Post-processing runtime summarization result...
INFO: Post-processing trace analysis report...
INFO: Displaying ./scorep_bt-mz_B_8x4_trace/trace.cubex...

                [GUI showing trace analysis report]
```

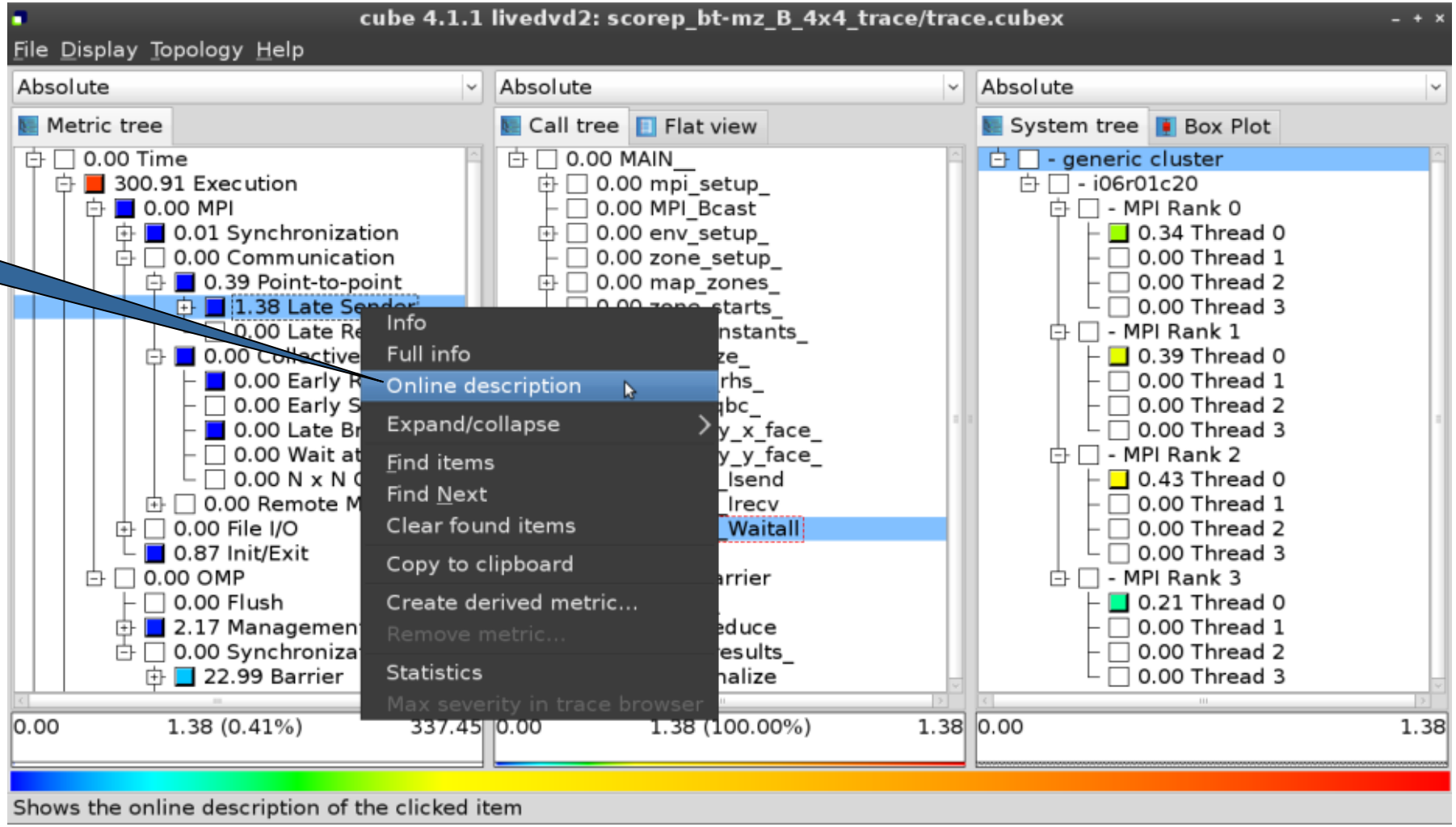
Post-processed trace analysis report

Additional trace-based metrics in metric hierarchy



Online metric description

Access online metric description via context menu



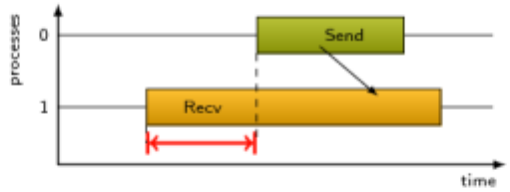
Online metric description

Performance properties

Late Sender Time

Description:

Refers to the time lost waiting caused by a blocking receive operation (e.g., `MPI_Recv` or `MPI_Wait`) that is posted earlier than the corresponding send operation.



If the receiving process is waiting for multiple messages to arrive (e.g., in an call to `MPI_Waitall`), the maximum waiting time is accounted, i.e., the waiting time due to the latest sender.

Unit:

Seconds

Diagnosis:

Try to replace `MPI_Recv` with a non-blocking receive `MPI_Irecv` that can be posted earlier, proceed concurrently with computation, and complete with a wait operation after the message is expected to have been sent. Try to post sends earlier, such that they are available when receivers need them. Note that outstanding messages (i.e., sent before the receiver is ready) will occupy internal message buffers, and that large numbers of posted receive buffers will also introduce message management overhead, therefore moderation is advisable.

Parent:

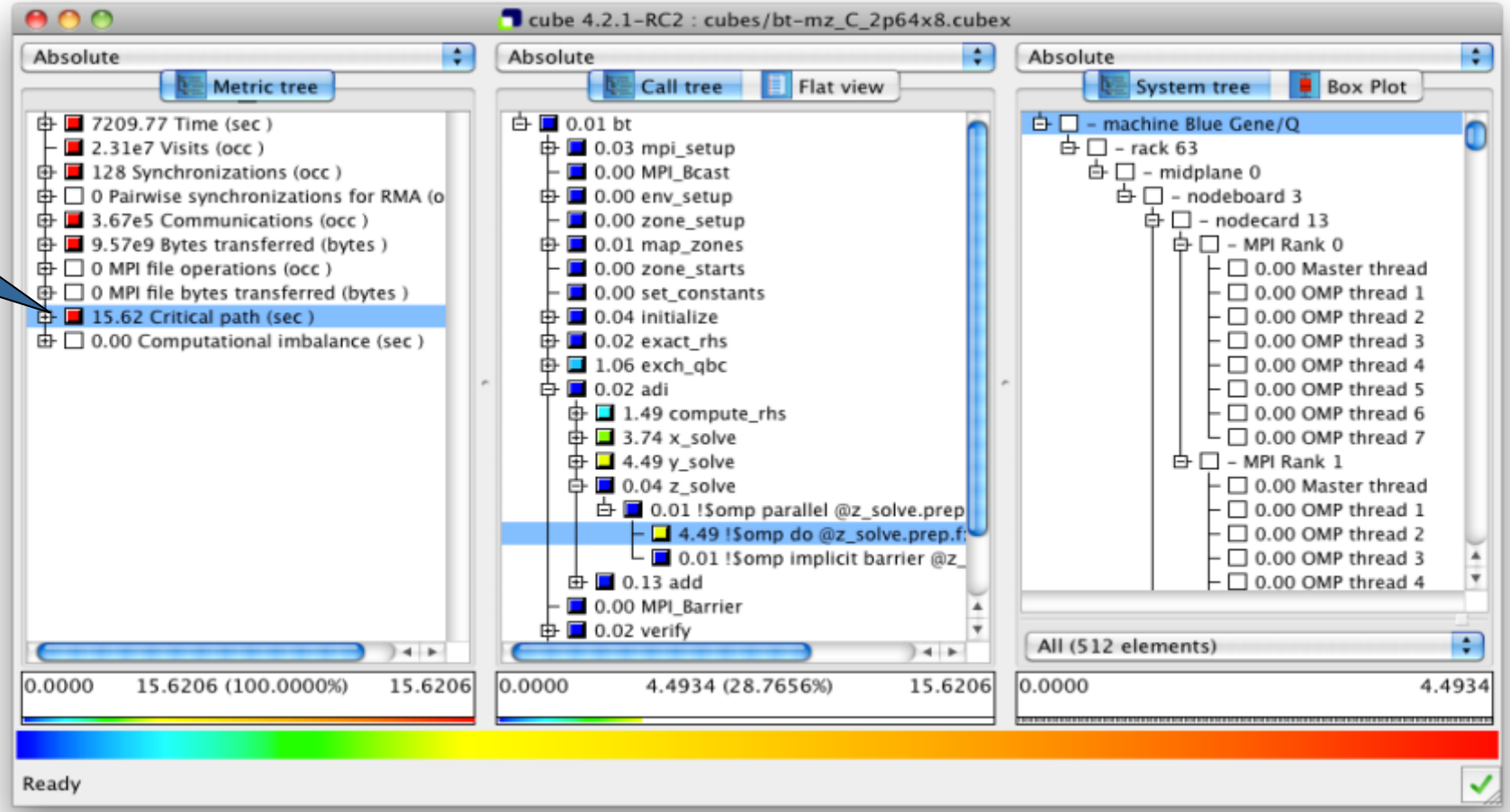
MPI Point-to-point Communication Time

Children:

Close

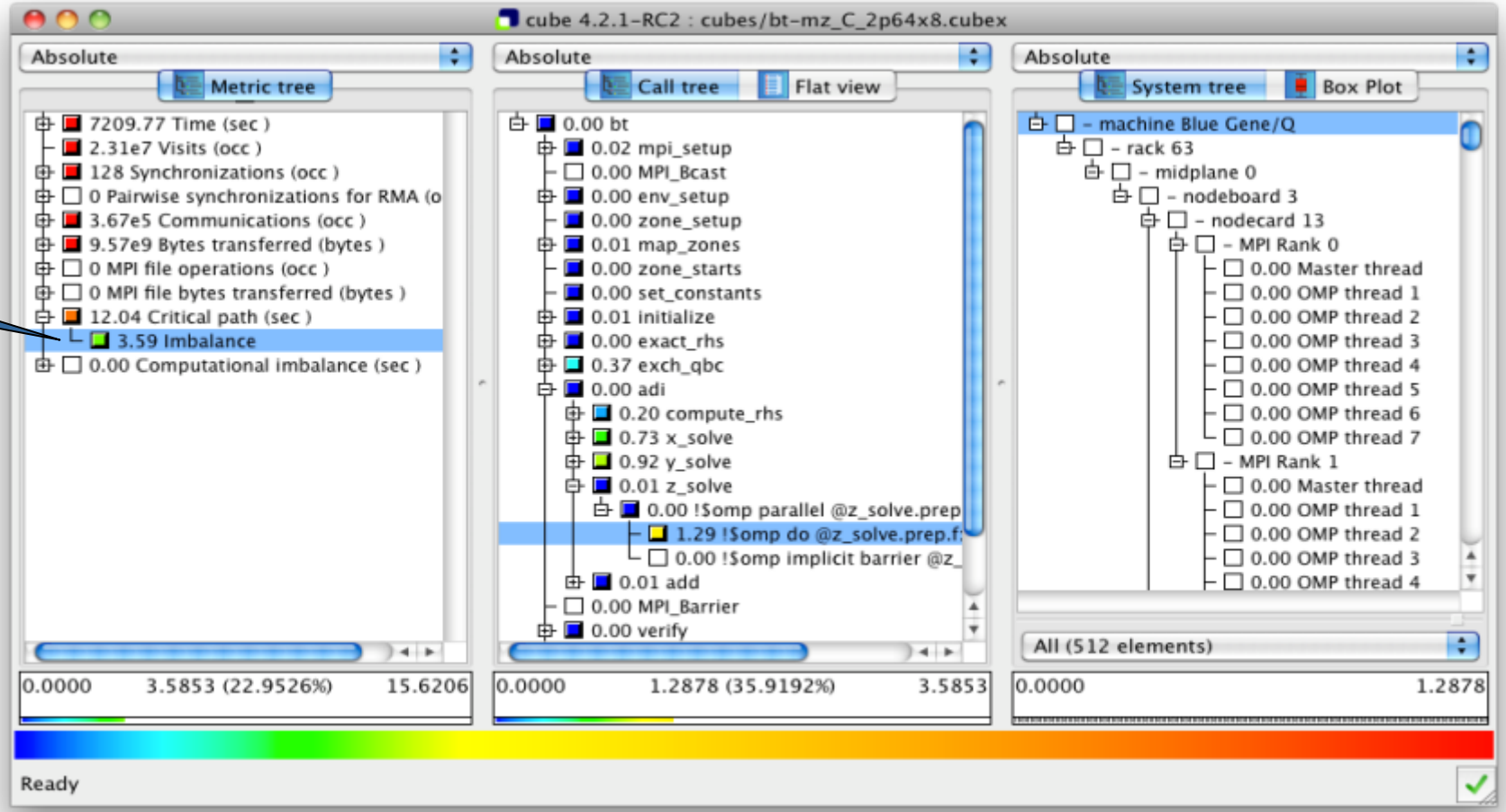
Critical-path analysis

Critical-path profile shows wall-clock time impact



Critical-path analysis

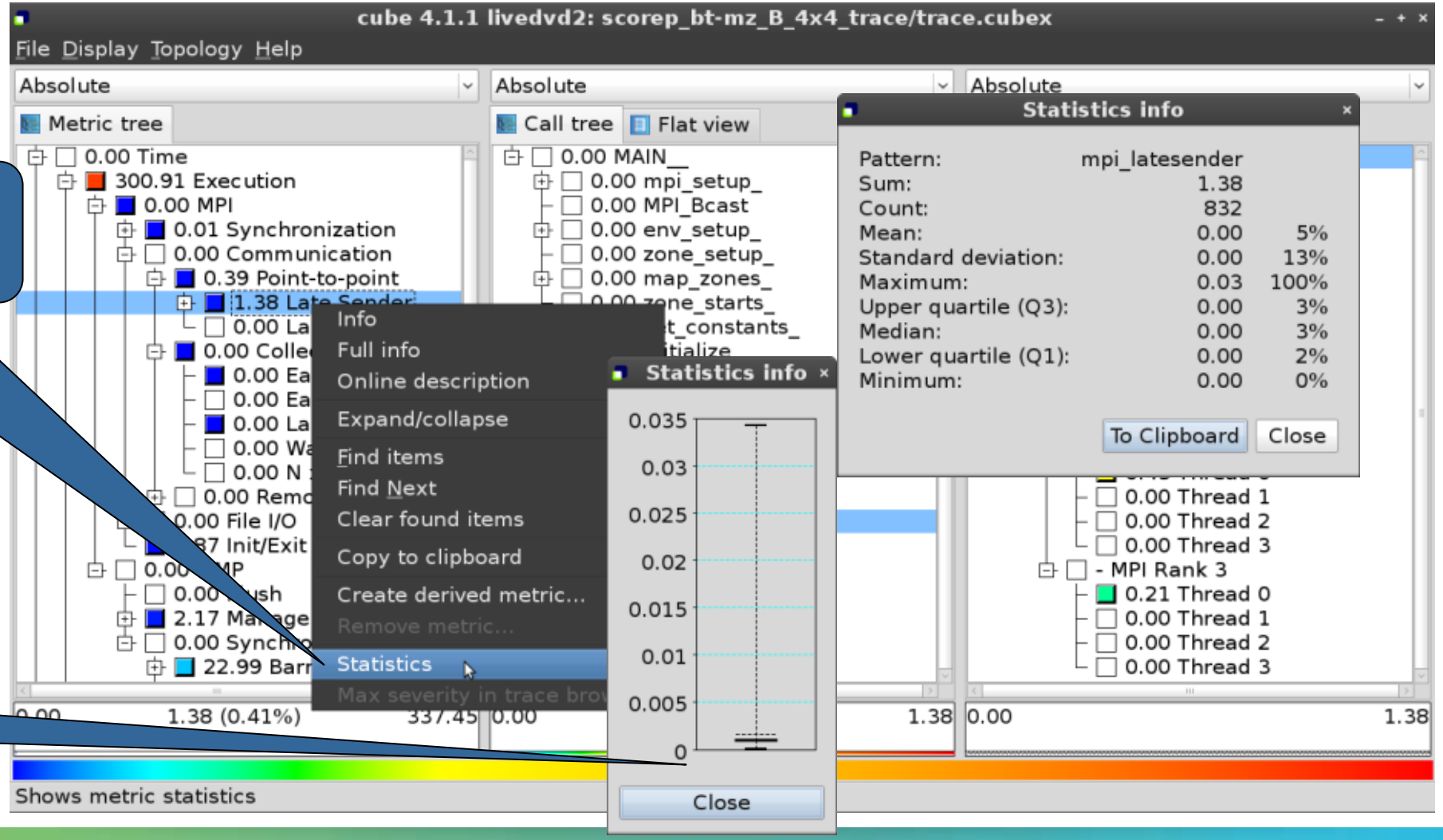
Critical-path imbalance highlights inefficient parallelism



Pattern instance statistics

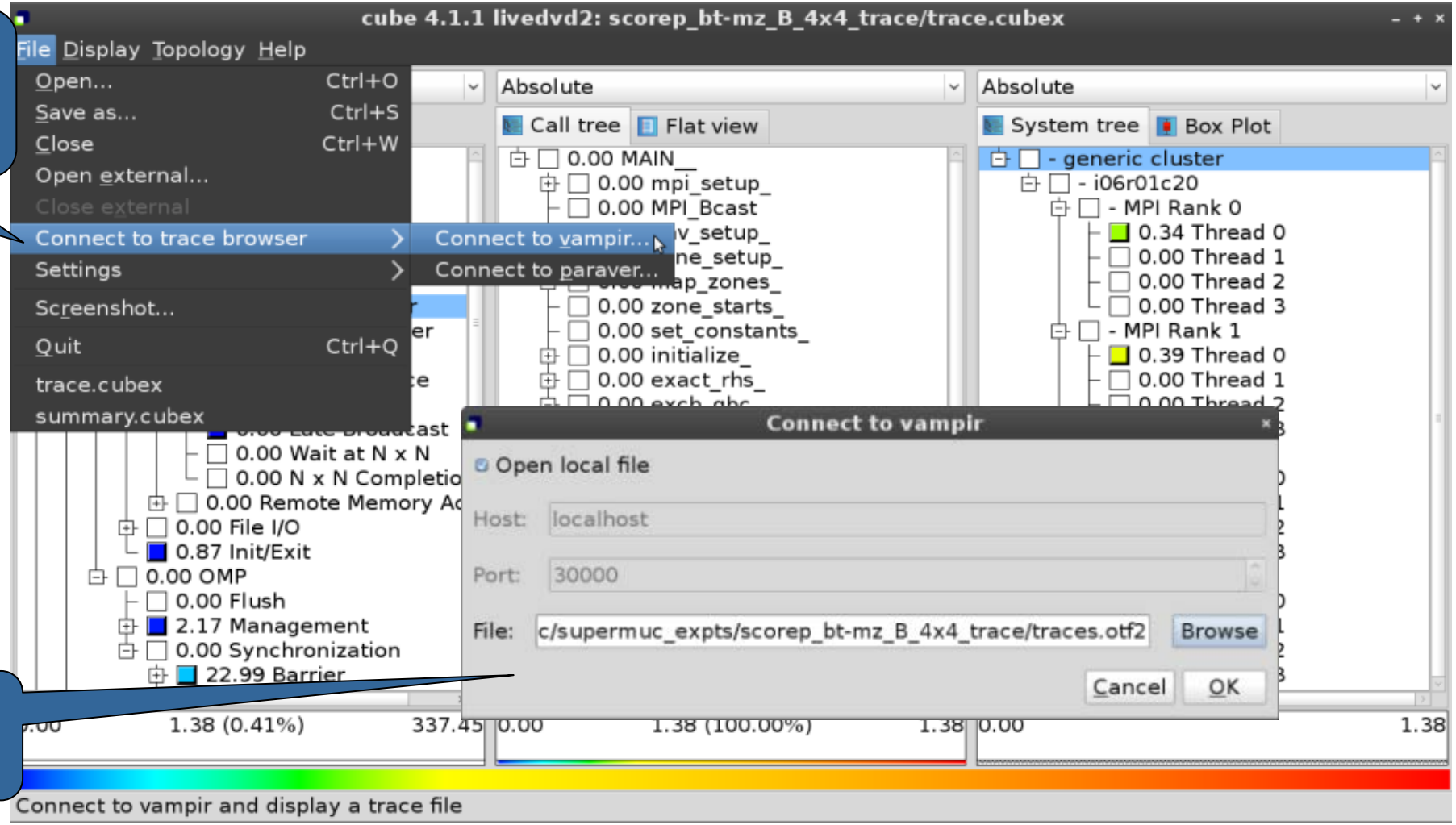
Access pattern instance statistics via context menu

Click to get statistics details



Connect to Vampir trace browser

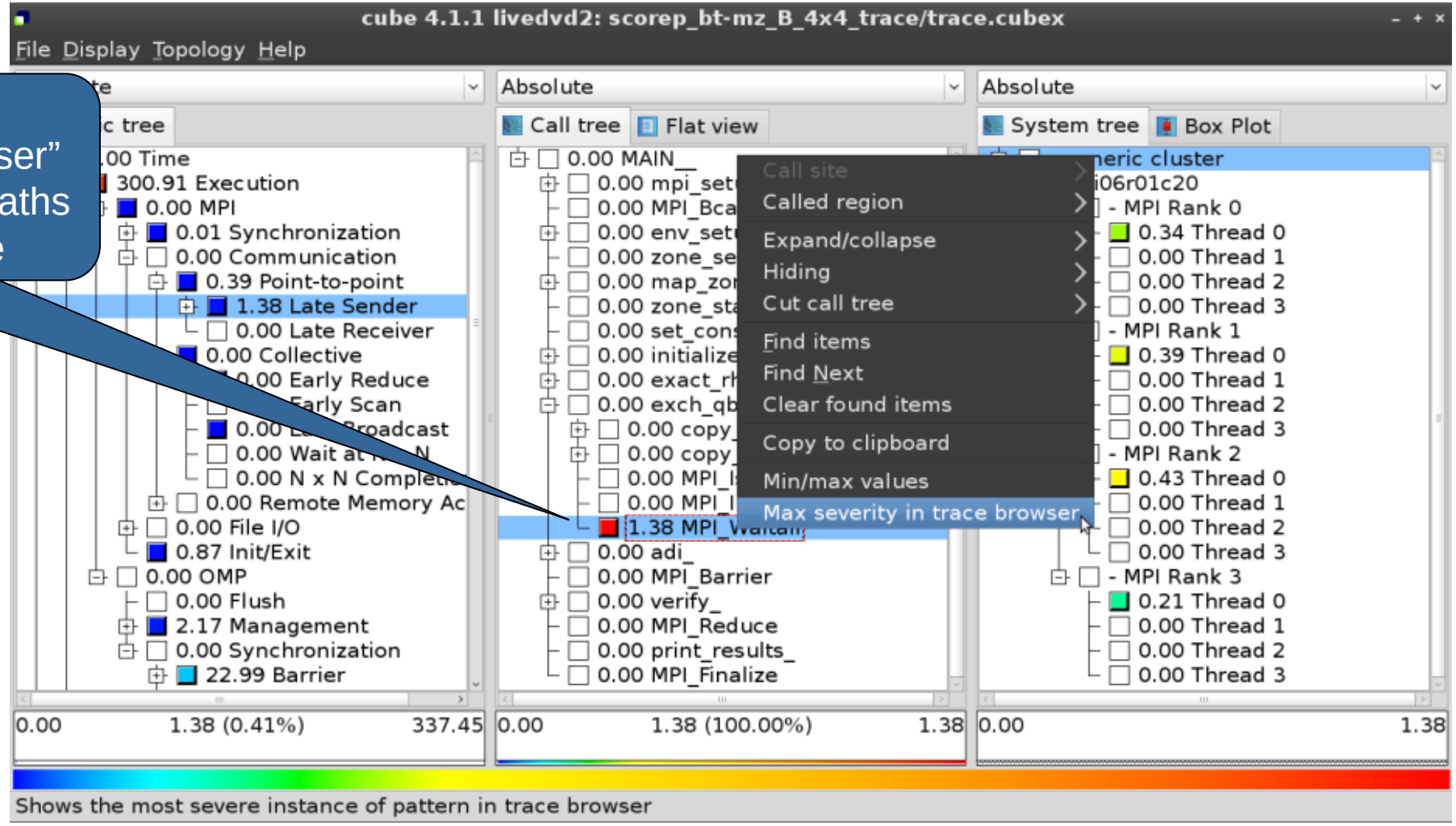
To investigate most severe pattern instances, connect to a trace browser…



…and select trace file from the experiment directory

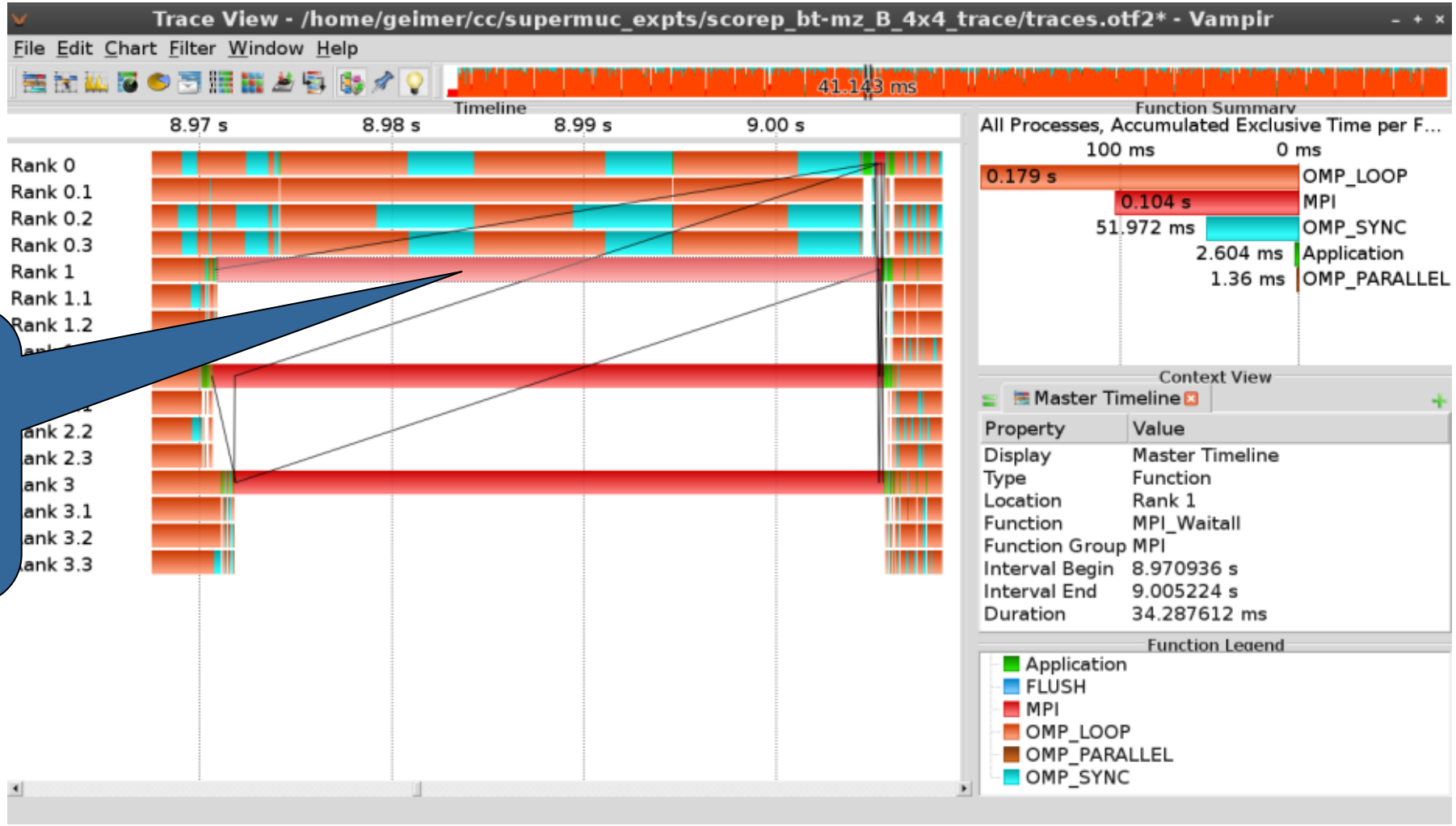
Show most severe pattern instances

Select
“Max severity in trace browser”
from context menu of call paths
marked with a red frame



Investigate most severe instance in Vampir

Vampir will automatically zoom to the worst instance in multiple steps (i.e., undo zoom provides more context)



Further information

Scalable performance **a**nalysis of **la**rge-**sc**ale parallel **a**pplications

- Toolset for scalable performance measurement & analysis of MPI, OpenMP & hybrid parallel applications
- Supporting most popular HPC computer systems
- Available under 3-clause BSD open-source license
- Sources, documentation & publications:
  - http://www.scalasca.org
  - mailto: scalasca@fz-juelich.de

scalasca