Automatic trace analysis with the Scalasca Trace Tools

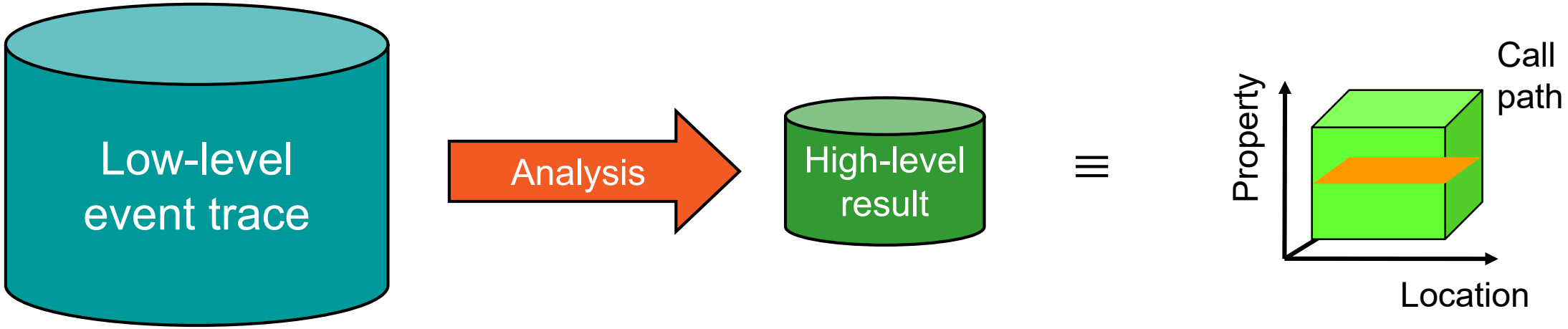
Radita Liem
RWTH Aachen University

(with content used with permission from tutorials by Markus Geimer & Brian Wylie, JSC)

trace tools
scalasca

Automatic trace analysis

- Idea
  - Automatic search for patterns of inefficient behaviour
  - Classification of behaviour & 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

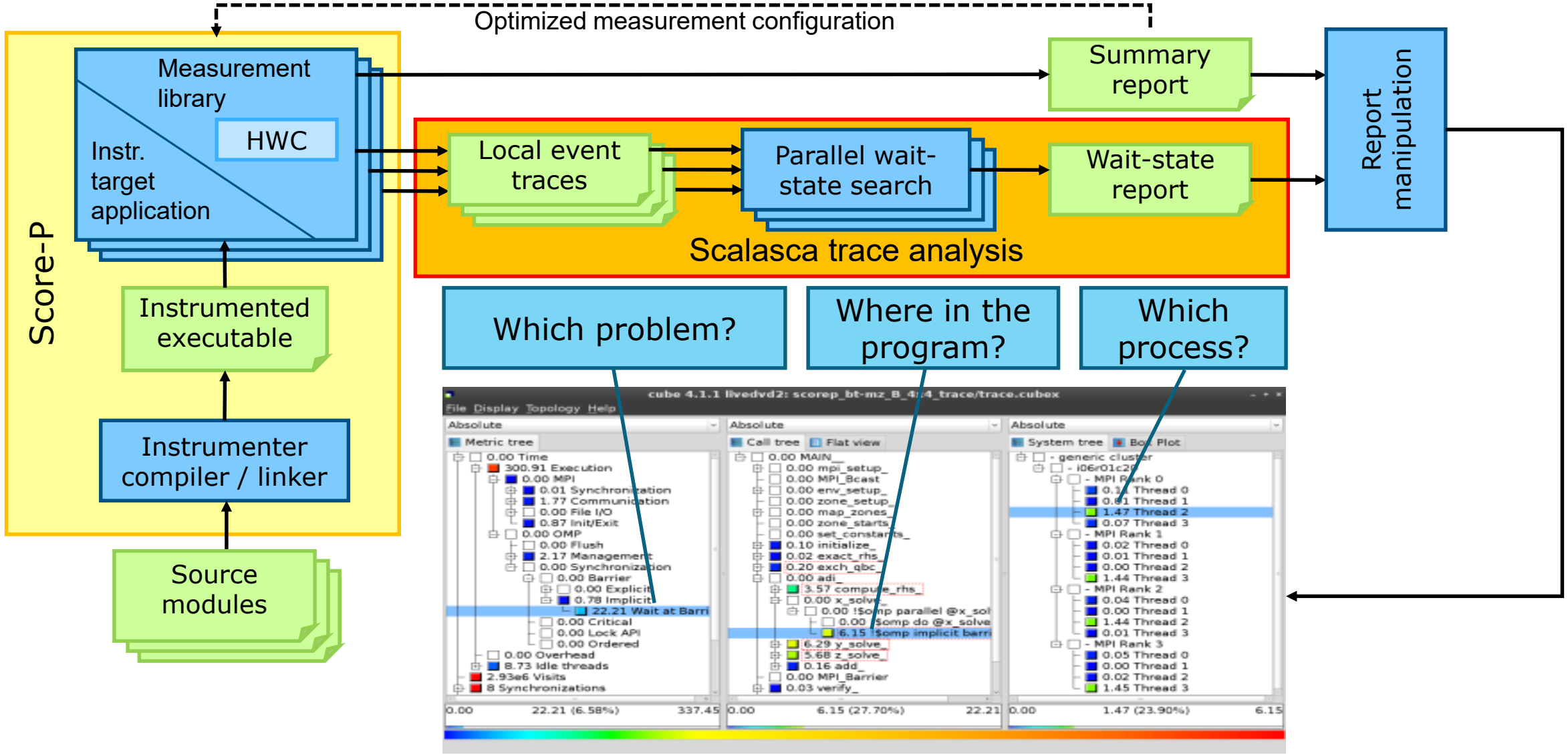
Scalasca Trace Tools: Objective

- Development of a **scalable trace-based** performance analysis toolset for the most popular parallel programming paradigms
  - Current focus: MPI, OpenMP, and (to a limited extend) POSIX threads
- 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 v2.5 coordinated with Score-P v5.0 (March 2019), also works with later versions
  - Pre-release version used for the workshop, v2.5 also available as fallback

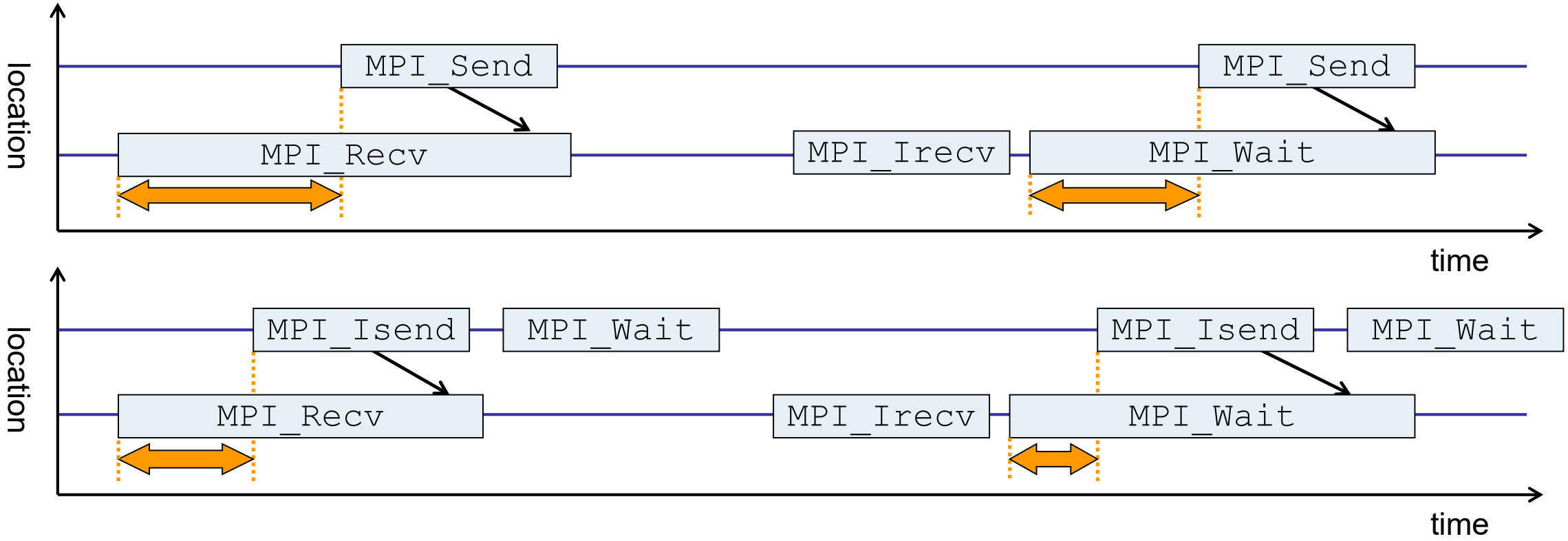
Scalasca Trace Tools: Features

- Open source, 3-clause BSD license
- Fairly portable
  - IBM Blue Gene, Cray XT/XE/XK/XC, SGI Altix, Fujitsu FX systems, Linux clusters (x86, Power, ARM), Intel Xeon Phi, ...
- Uses Score-P instrumenter & measurement libraries
  - Scalasca v2 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
    - with MPI thread level exceeding MPI_THREAD_FUNNELED
    - containing Memory events, CUDA/OpenCL device events (kernel, memcpy), SHMEM, or OpenMP nested parallelism
  - PAPI/rusage metrics for trace events are ignored

Scalasca workflow

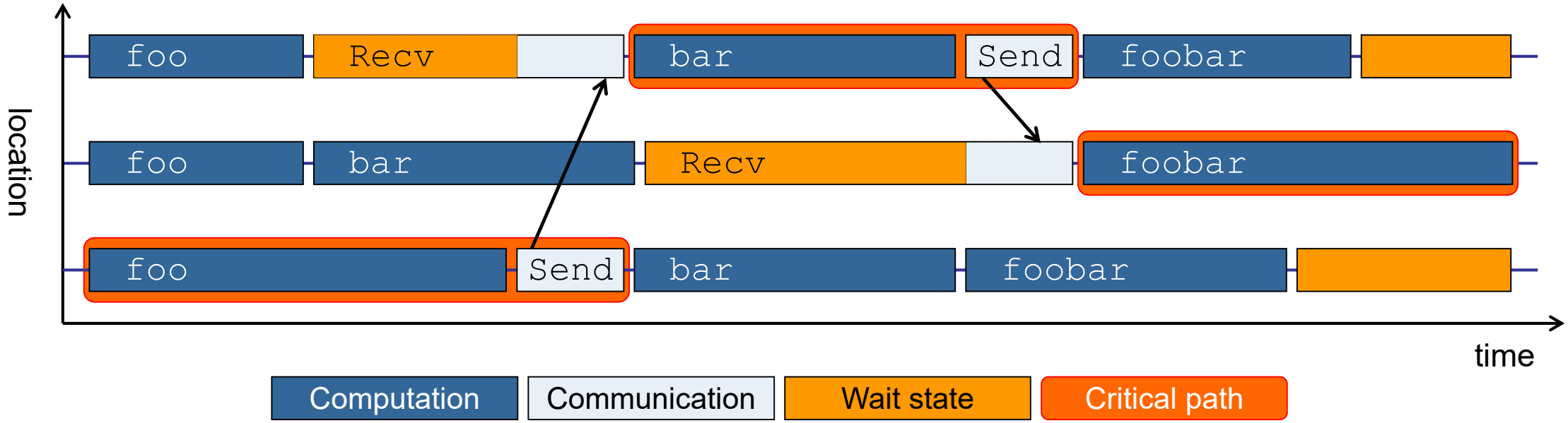


Example: "*Late Sender*" wait state



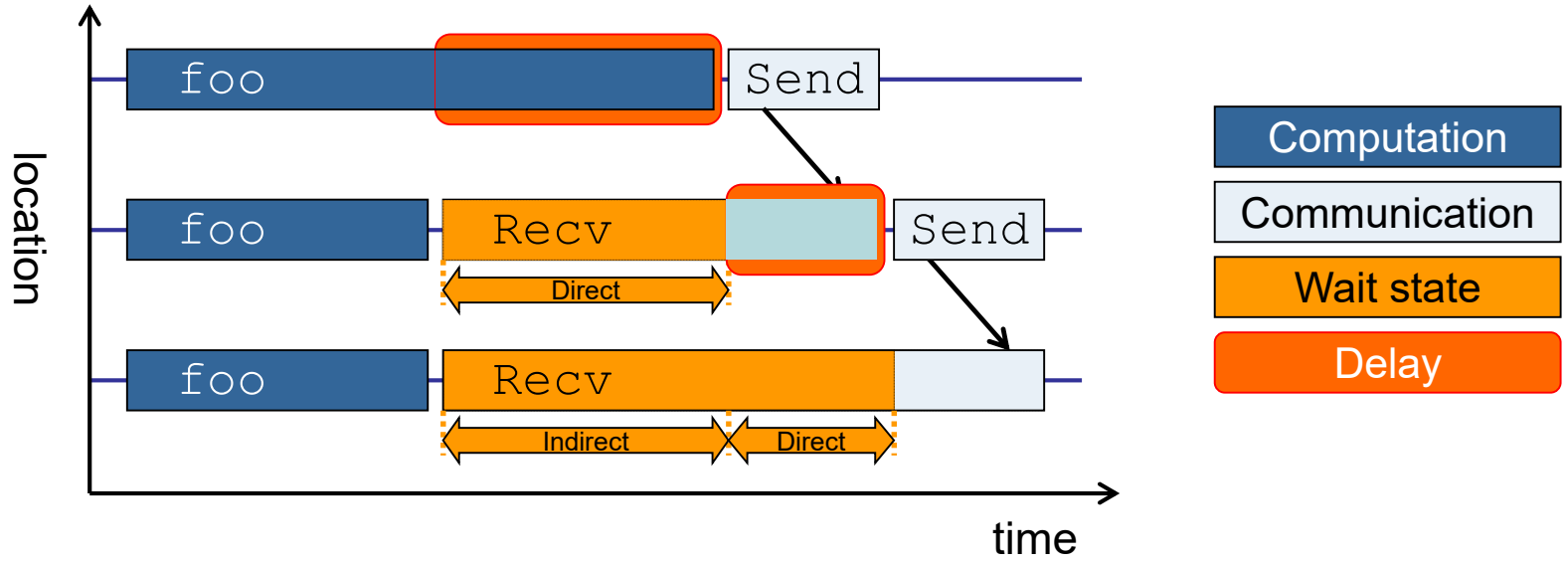
- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication

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

---

trace tools
scalasca

Loading Scalasca into the Environment

- Welcome to the 2nd day! You need to reload the necessary modules to the environment

```
# load dependency modules to the environment
% module load Stages/2022 # if you are in Jupyter Xpra
% module load GCC ParaStationMPI

# load scalasca
% module load Scalasca/2.6

% module list
Currently Loaded Modules:
  1) Stages/2022              (S)    21) bzip2/.1.0.8         (H)   41) libunwind/.1.5.0     (H)
  2) GCCcore/.11.2.0          (H)    22) PCRE/.8.45           (H)   42) OpenGL/2021b         (g)
  3) zlib/.1.2.11             (H)    23) util-linux/.2.37     (H)   43) NASM/.2.15.05        (H)
[...]
 14) libxml2/.2.9.10          (H)    34) DBus/.1.13.18        (H)   54) OTF2/.2.3            (H)
 15) mpi-settings/UCX                35) OpenSSL/1.1                55) OPARI2/.2.0.6        (H)
 16) ParaStationMPI/5.5.0-1   (g)    36) libevent/.2.1.12     (H)   56) PAPI/6.0.0.1
 17) double-conversion/3.1.6         37) GMP/6.2.1                  57) PDT/.3.25.1          (H)
 18) libffi/.3.4.2            (H)    38) nettle/.3.7.3        (H)   58) Score-P/7.1
 19) ncurses/.6.2             (H)    39) libdrm/.2.4.108      (H)   59) Scalasca/2.6
 20) gettext/.0.21            (H)    40) LLVM/13.0.0
```

Scalasca command – One command for (almost) everything

```
% scalasca
Scalasca 2.6
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 convenience command: scan / scalasca -analyze

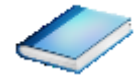
```
% scan
Scalasca 2.6: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
      where {options} may include:
  -h    Help      : show this brief usage message and exit.
  -v    Verbose   : increase verbosity.
  -n    Preview   : show command(s) 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.
  -R #runs        : Specify the number of measurement runs per config.
  -M cfgfile      : Specify a config file for a multi-run measurement.
  -P preset       : Specify a preset for a multi-run measurement, e.g., 'pop'.
  -L              : List available multi-run presets.
  -D cfgfile      : Check a multi-run config file for validity and dump
                  : the processed configuration for comparison.
```

- Scalasca measurement collection & analysis nexus

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)

Scalasca advanced command: scout - Scalasca automatic trace analyzer



```
% scout.hyb --help
SCOUT   (Scalasca 2.6)
Copyright (c) 1998-2021 Forschungszentrum Juelich GmbH
Copyright (c) 2014-2021 RWTH Aachen University
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 convenience command: square / scalasca -examine

```
% square
Scalasca 2.6: analysis report explorer
usage: square [OPTIONS] <experiment archive | cube file>
   -C <none | quick | full> : Level of sanity checks for newly created reports
   -c <number>              : Consider number of counters when doing scoring (-s)
   -F                       : Force remapping of already existing reports
   -f filtfile              : Use specified filter file when doing scoring (-s)
   -s                       : Skip display and output textual score report
   -v                       : Enable verbose mode
   -n                       : Do not include idle thread metric
   -S <mean | merge>        : Aggregation method for summarization results of
                              each configuration (default: merge)
   -T <mean | merge>        : Aggregation method for trace analysis results of
                              each configuration (default: merge)
   -A                       : Post-process every step of a multi-run experiment
   -I                       : Ignore structural sanity checks and force aggregation
                              of measurements in a multi-run experiment
   -x <scorep-score opt>    : Pass option(s) to scorep-score
```

- Scalasca analysis report explorer (Cube)

Recap: Local installation (JUSUF)

- Load the Scalasca module

```
# load dependency modules to the environment
% module load Stages/2022 # if you are in Jupyter Xpra
% module load GCC ParaStationMPI

# load scalasca
% module load Scalasca/2.6
```

- If you haven't done this in the day 1, copy the BT-MZ to your own work directory

```
% cd work/[your_JUSUF_username] #created from 'source setup.sh
% cp /p/project/training2214/NPB3.3-MZ-MPI.tar.gz .
% tar -zxvf NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI
```

BT-MZ summary measurement collection...

```
% cd bin.scorep
% cp ../jobscript/jusuf/scalasca.sbatch .
% cat scalasca.sbatch

# Benchmark configuration (disable load balancing with threads)
export NPB_MZ_BLOAD=0
PROCS=8
CLASS=C

# Measurement configuration
export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_TOTAL_MEMORY=80M
#export SCAN_ANALYZE_OPTS="--time-correct"

# Run the application
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
scalasca -analyze  srun  ./bt-mz_$CLASS.$PROCS
```

- Change to directory with the Score-P instrumented executable and edit the job script

Hint:
scan = scalasca -analyze
-s = profile/summary (def)

```
% sbatch scalasca.sbatch
```

- Submit the job

BT-MZ summary measurement

```
S=C=A=N: Scalasca 2.6 runtime summarization
S=C=A=N: ./scorep_bt-mz_C_8x6_sum experiment archive
S=C=A=N: Wed May 18 11:19:04 2022: Collect start
/usr/bin/srun ./bt-mz_C.8


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

 Number of zones:  16 x  16
 Iterations: 200    dt:   0.000100
 Number of active processes:     8

 [... More application output ...]

S=C=A=N: Wed May 18 11:19:19 2022: Collect done (status=0) 15s
S=C=A=N: ./scorep_bt-mz_C_8x6_sum complete.
```

- Run the application using the Scalasca measurement collection & analysis nexus prefixed to launch command
- Creates experiment directory: **scorep_bt-mz_C_8x6_sum**

BT-MZ summary analysis report examination

- Score summary analysis report

```
% square -s  scorep_bt-mz_C_8x6_sum
INFO: Post-processing runtime summarization report (profile.cubex)...
/p/software/jusuf/stages/2022/software/Score-P/7.1-gpsmpi-2021b/bin/
scorep-score  -r ./scorep_bt-mz_C_8x6_sum/profile.cubex > ./scorep_bt-mz_C_8x6_sum/scorep.score
INFO: Score report written to ./scorep_bt-mz_C_8x6_sum/scorep.score
```

- Post-processing and interactive exploration with Cube

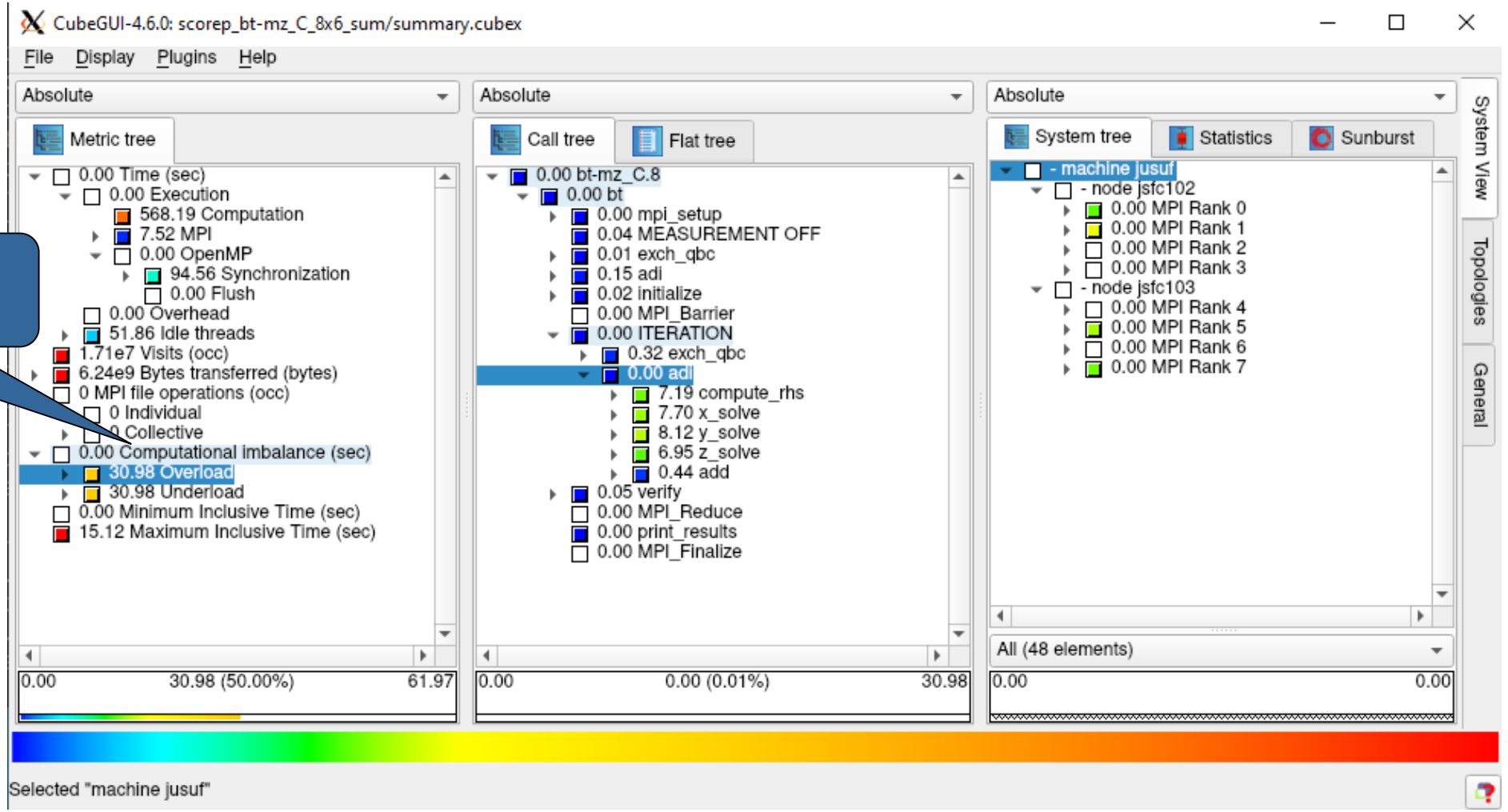
```
% square scorep_bt-mz_C_8x6_sum
INFO: Post-processing runtime summarization report (profile.cubex)...

                              [GUI showing summary analysis report]
```

- The post-processing derives additional metrics and generates a structured metric hierarchy

Post-processed summary analysis report

Split base metrics into more specific metrics



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...

```
% vi scalasca.sbatch

# Benchmark configuration (disable load balancing with threads)
export NPB_MZ_BLOAD=0
PROCS=8
CLASS=C

# Measurement configuration
export SCOREP_FILTERING_FILE=../config/scorep.filt
export SCOREP_TOTAL_MEMORY=80M
#export SCAN_ANALYZE_OPTS="--time-correct"

# Run the application
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
scalasca -analyze -t srun  ./bt-mz_$CLASS.$PROCS
```

- Change to directory with the Score-P instrumented executable and edit the job script
- **Add "-t" to the `scan` command**

```
% sbatch scalasca.sbatch
```

- **Submit the job**

BT-MZ trace measurement … collection

```
S=C=A=N: Scalasca 2.6 trace collection and analysis
S=C=A=N: ./scorep_bt-mz_C_8x6_trace experiment archive
S=C=A=N: Wed May 18 11:34:36 2022: Collect start
/usr/bin/srun ./bt-mz_C.8


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

 Number of zones:  16 x  16
 Iterations: 200    dt:   0.000100
 Number of active processes:     8

 Use the default load factors with threads
 Total number of threads:     48   (  6.0 threads/process)

 Calculated speedup =     47.97

 [... More application output ...]
```

- Starts measurement with collection of trace files …

BT-MZ trace measurement … analysis

```
S=C=A=N: Wed May 18 11:34:56 2022: Collect done (status=0) 20s
S=C=A=N: Wed May 18 11:34:56 2022: Analyze start
/usr/bin/srun /p/software/jusuf/stages/2022/software/Scalasca/2.6-gpsmpi-
2021b/bin/scout.hyb ./scorep_bt-mz_C_8x6_trace/traces.otf2
SCOUT   (Scalasca 2.6)
[..]
Analyzing experiment archive ./scorep_bt-mz_C_8x6_trace/traces.otf2

Opening experiment archive ... done (0.007s).
Reading definition data    ... done (0.009s).
Reading event trace data   ... done (0.207s).
Preprocessing              ... done (0.322s).
Analyzing trace data       ... done (7.428s).
Writing analysis report    ... done (0.309s).

Max. memory usage         : 869.758MB

Total processing time     : 8.443s
S=C=A=N: Wed May 18 11:35:05 2022: Analyze done (status=0) 9s
S=C=A=N: ./scorep_bt-mz_C_8x6_trace complete.
```

- Continues with automatic (parallel) analysis of trace files

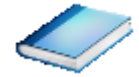
BT-MZ trace analysis report exploration

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

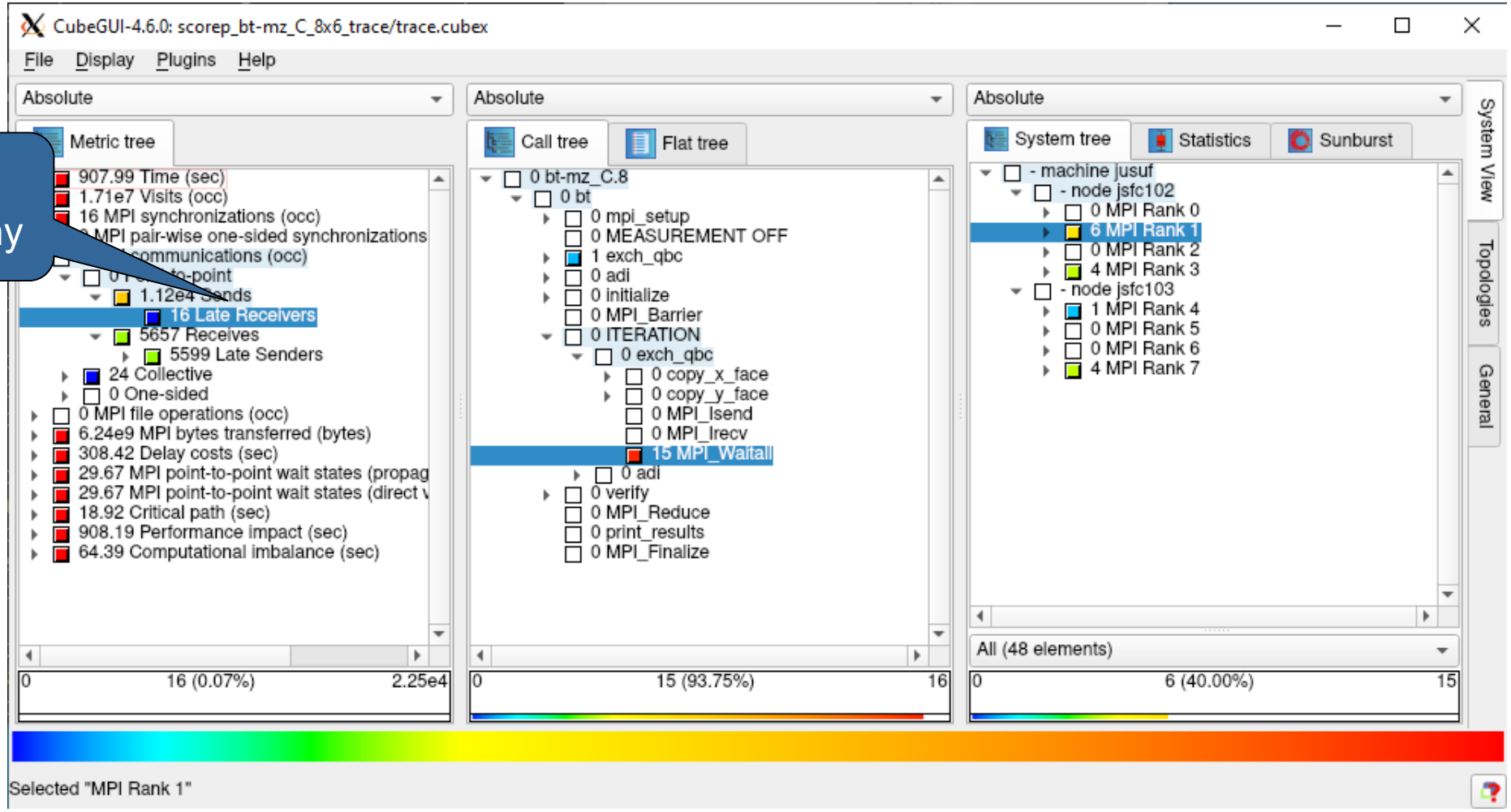
```
% square  scorep_bt-mz_C_8x6_trace
INFO: Post-processing runtime summarization report (profile.cubex)...
INFO: Post-processing trace analysis report (scout.cubex)...
INFO: Displaying ./scorep_bt-mz_C_8x6_trace/trace.cubex...

                    [GUI showing trace analysis report]
```

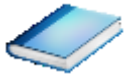
Post-processed trace analysis report



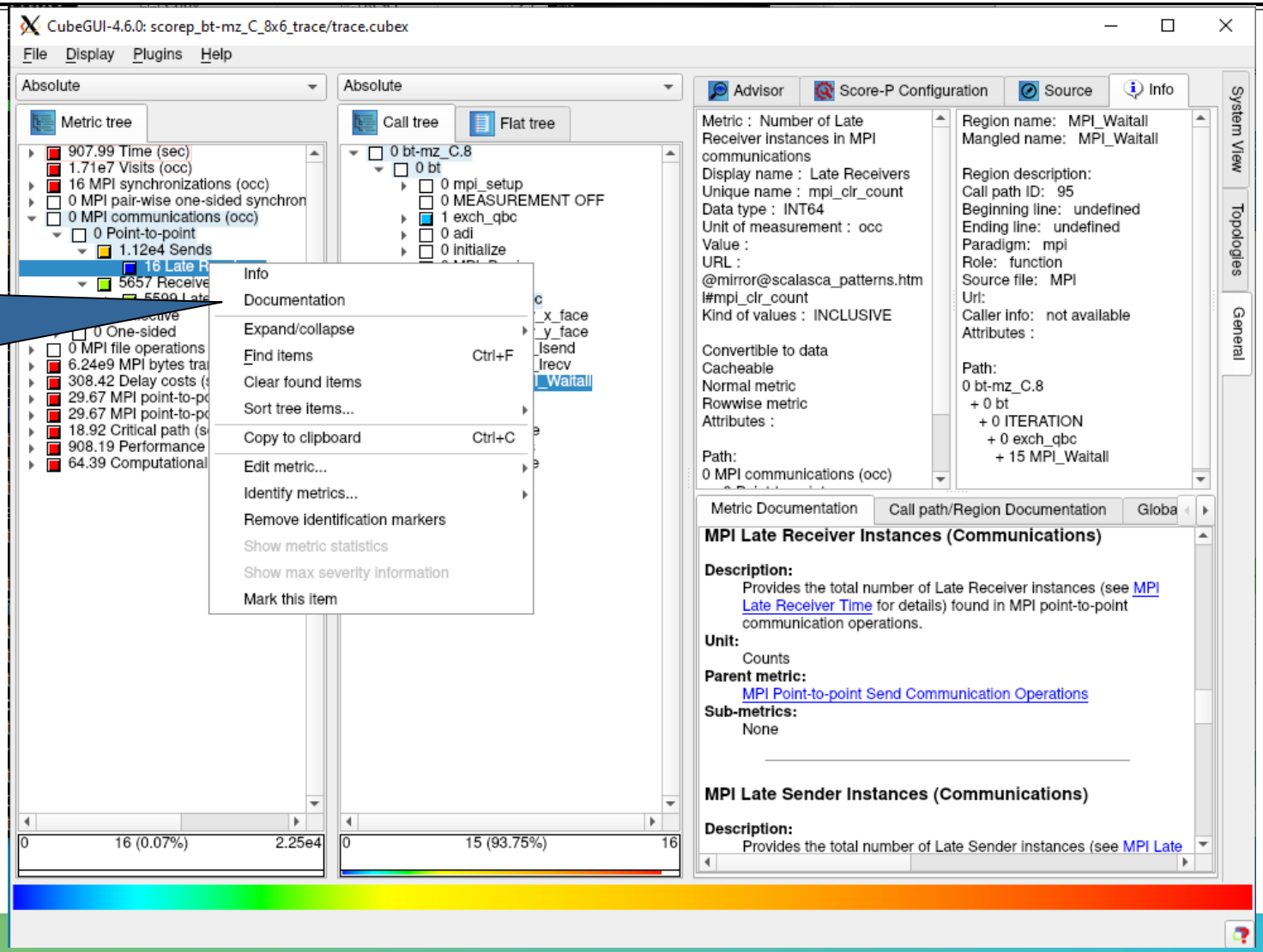
Additional trace-based metrics in metric hierarchy



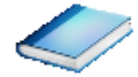
Online metric description



Documentation on the metrics will be displayed on the right hand side

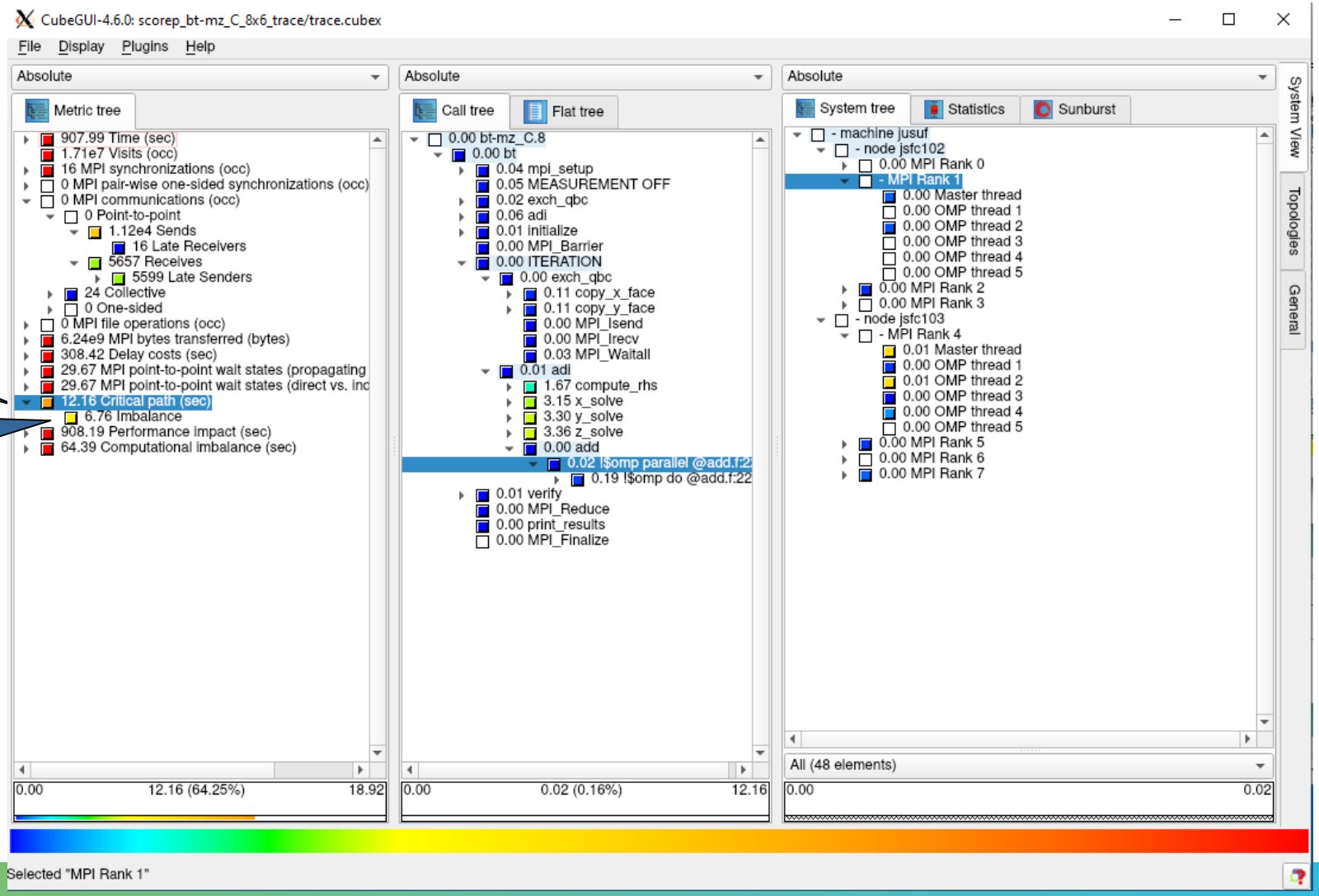


Critical-path analysis



Critical-path profile shows wall-clock time impact

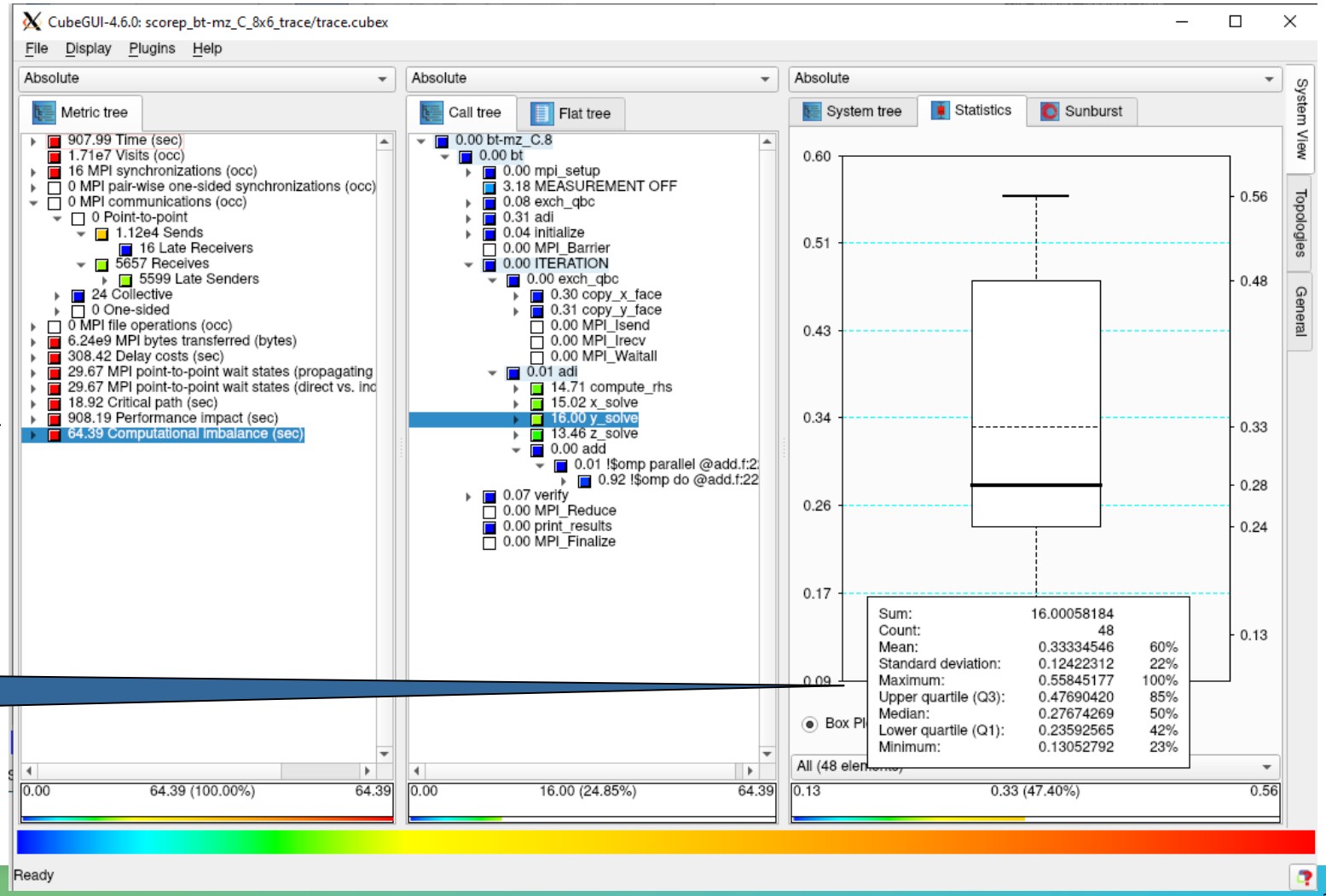
Critical-path imbalance highlights inefficient parallelism



Pattern instance statistics

Statistics of is available on the right hand side

Click to get statistics details



Exercises
(if you don't have your own code)

Warm-up

- Build the BT-MZ example code for class (i.e., problem size) “D”
  - Perform a baseline measurement w/o instrumentation
  - Re-build the executable with Score-P instrumentation
- Repeat the hands-on exercise with the new executable
  - Perform a summary measurement
  - Score the summary measurement result
  - Adjust the measurement configuration appropriately
  - Perform a trace measurement and analysis

Trace analysis report examination

- What is the poportion of computation time vs. parallelization overheads?
- Which code regions are mostly responsible for the overall execution time?
- Are there any load balancing issues?
- If so, in which routines?
- What are the most significant wait states/parallelization overheads?
- What are their root causes?