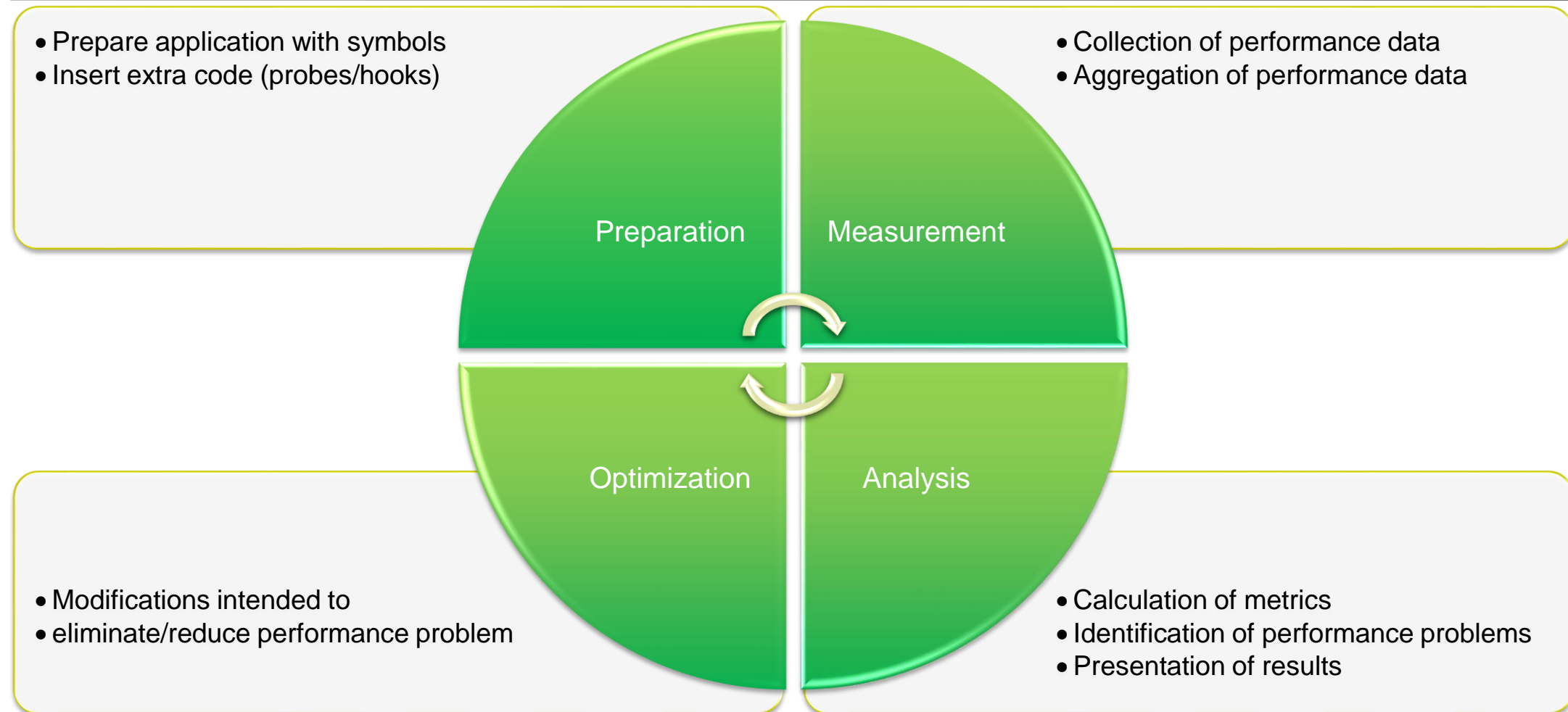


## Measurement with Score-P

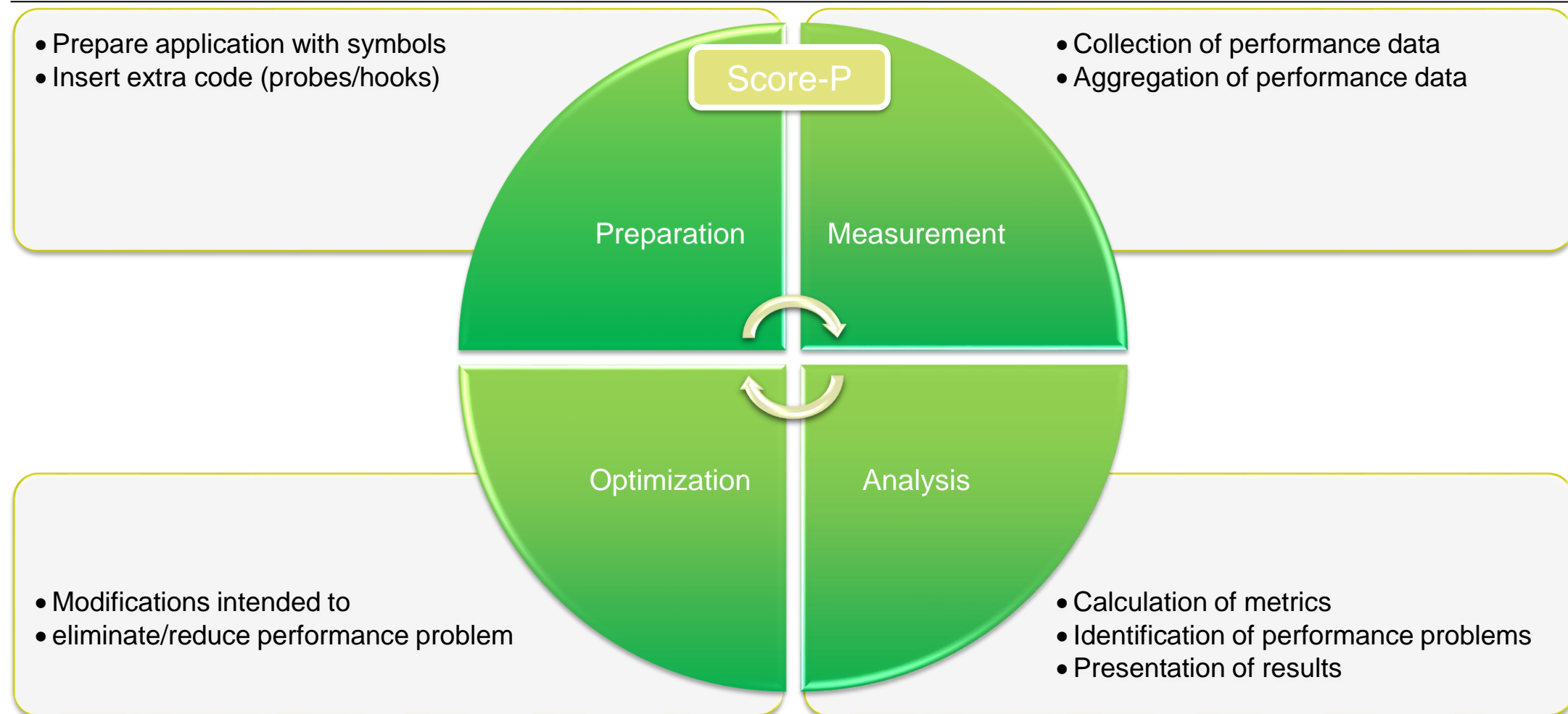
Marc Schlütter, JSC



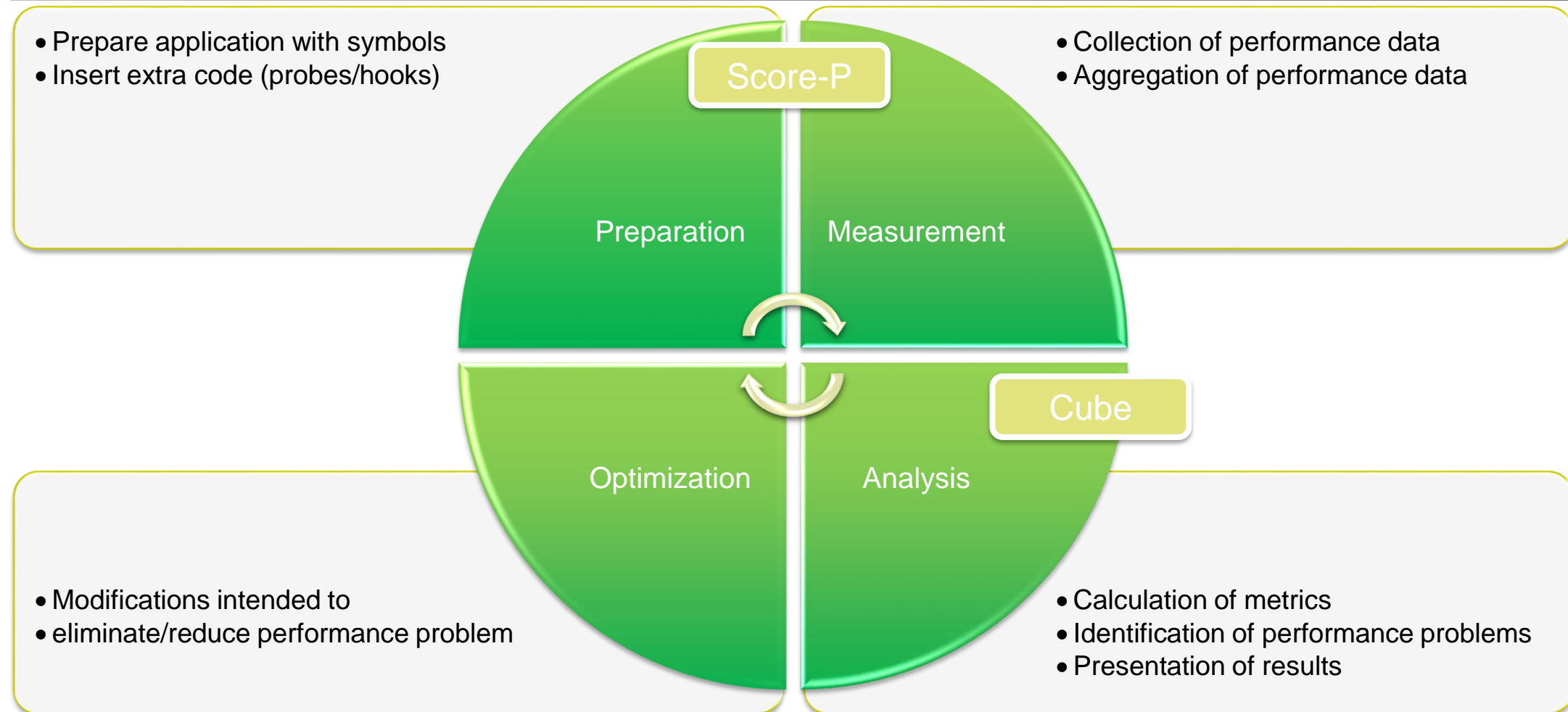
# Performance engineering workflow



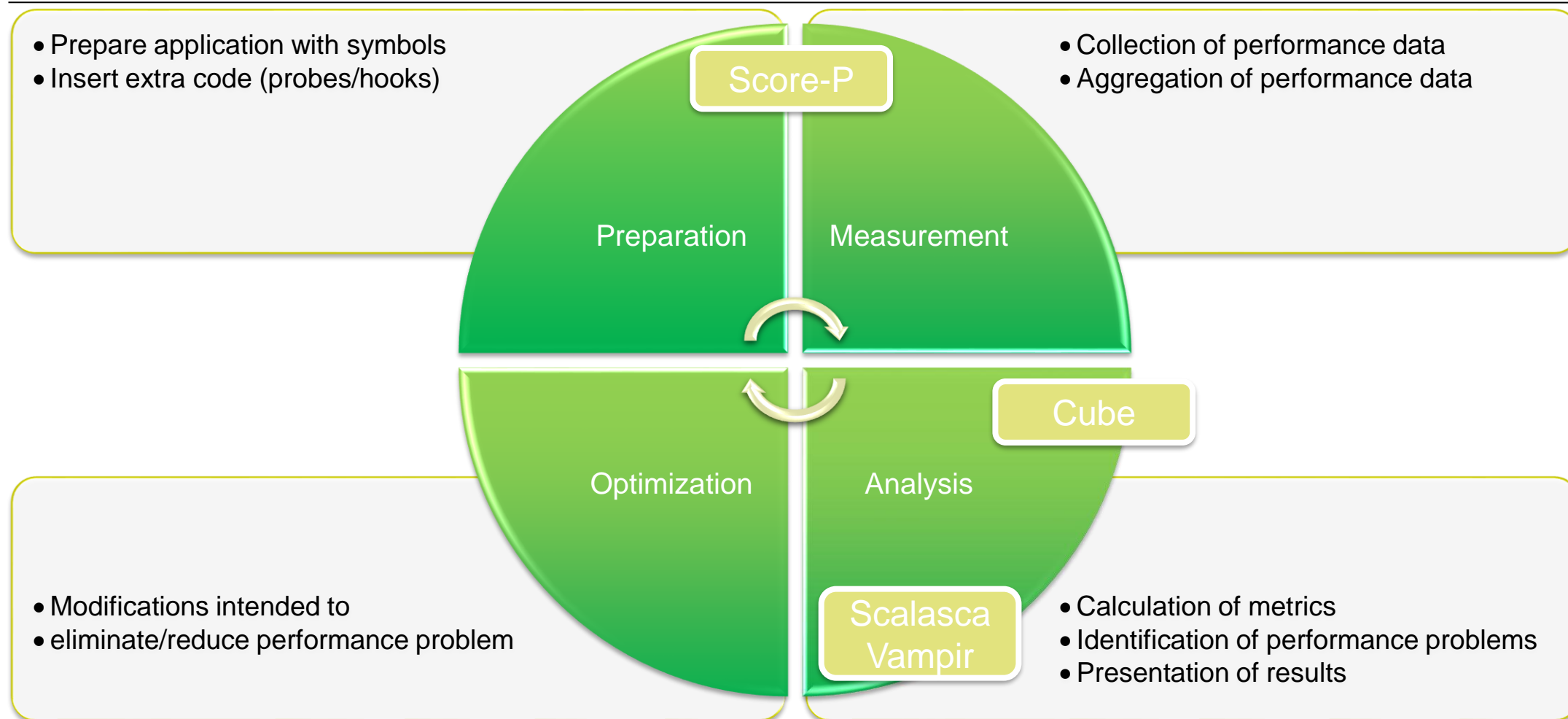
# Performance engineering workflow



# Performance engineering workflow



# Performance engineering workflow



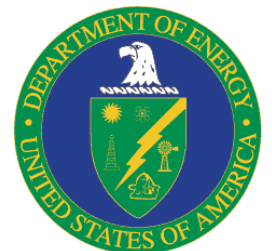
# Score-P

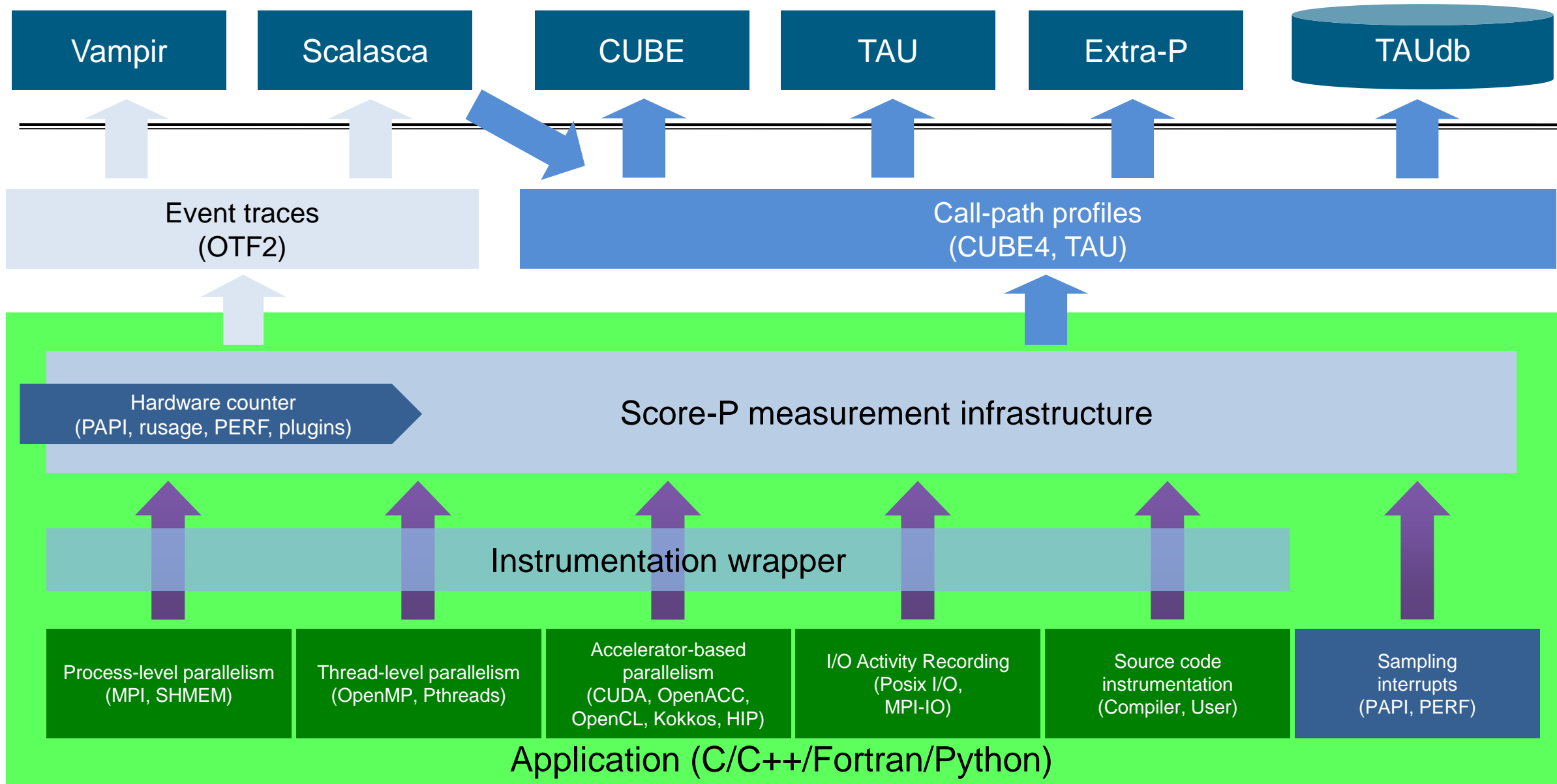
- Infrastructure for instrumentation and performance measurements
- Instrumented application can be used to produce several results:
  - Call-path profiling: CUBE4 data format used for data exchange
  - Event-based tracing: OTF2 data format used for data exchange
- Supported parallel paradigms:
  - Multi-process: MPI, SHMEM
  - Thread-parallel: OpenMP, Pthreads
  - Accelerator-based: CUDA, ROCm, OpenCL, OpenACC, Kokkos
- Open Source; portable and scalable to all major HPC systems
- Initial project funded by BMBF
- Close collaboration with PRIMA project funded by DOE

GEFÖRDERT VOM



Bundesministerium  
für Bildung  
und Forschung



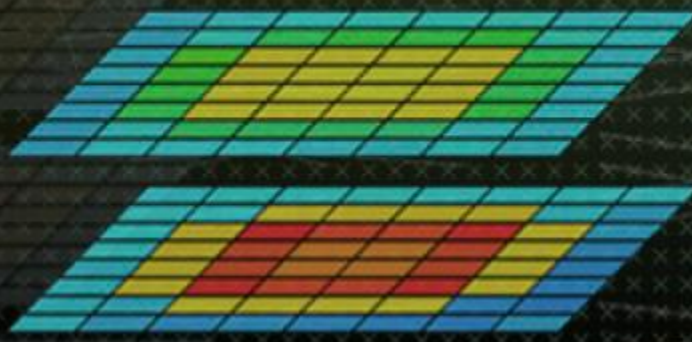


# Partners

---

- Forschungszentrum Jülich, Germany
- Gesellschaft für numerische Simulation mbH Braunschweig, Germany
- RWTH Aachen, Germany
- Technische Universität Darmstadt, Germany
- Technische Universität Dresden, Germany
- Technische Universität München, Germany
- University of Oregon, Eugene, USA





## Demo: NPB-MZ-MPI / BT

---

# Performance analysis workflow: Reference Run

Reference Build & Run

Preparation

Reference Build & Run

- Getting uninstrumented and undisturbed reference execution times
- Verifying a working combination of application and system configuration

# Performance analysis workflow: Application Instrumentation

Reference Build & Run



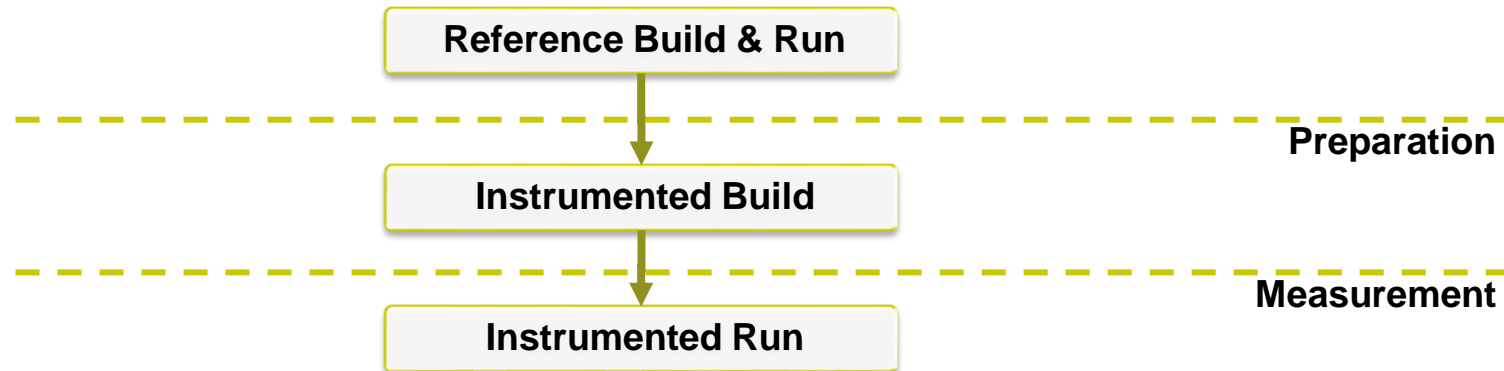
Instrumented Build

Preparation

## Instrumented Build

- First instrumentation of the code with Score-P
- Default usage: prefix every compiler call with `scorep`
- Automatic recognition of paradigms in most cases
- Explicit options available in case these fail

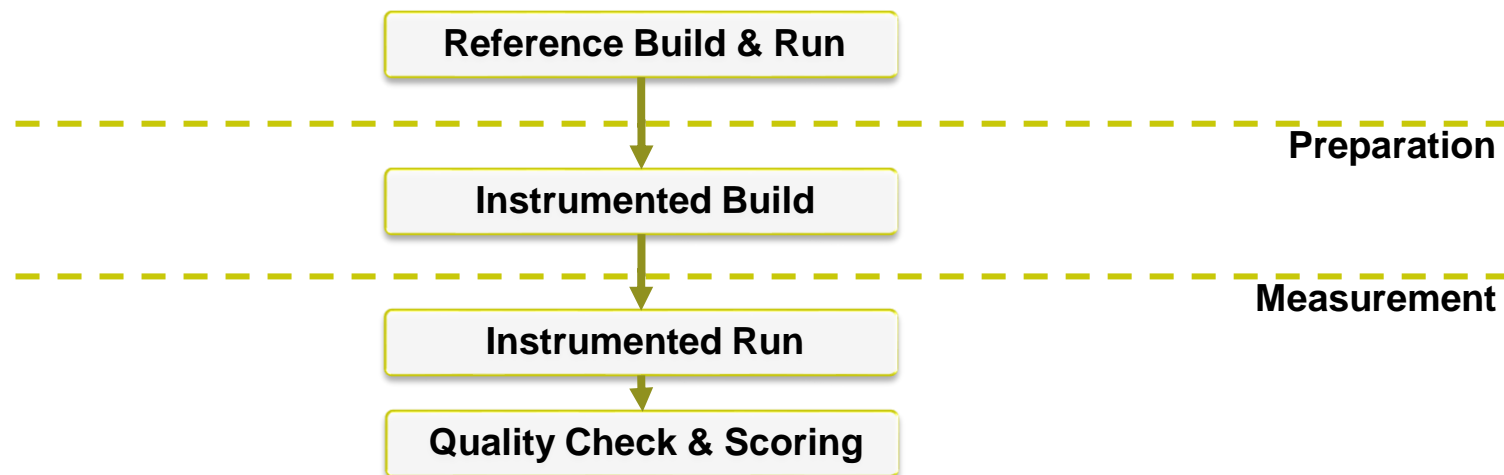
# Performance analysis workflow: Instrumented Run



## Instrumented Run

- Run possible using the same execution cmds/scripts using Score-P default settings
- Controlled via environment variables
- `scorep-info config-vars` for a list of all variables
- Measurement directory contains the used configuration
- Using the reference run to compare runtime overhead

# Performance analysis workflow: Quality Check & Scoring



## Quality Check & Scoring

- Using `scorep-score` to evaluate sources of overhead in the execution
- Runtime overhead related to time per visit and visits as most of Score-P's overhead is static per visit
- Information about expected trace sizes for later measurements, a direct function of the number of visits for a region
- `scorep-score -r` : listing per region(functions mostly)
- `scorep-score -g`: generating an initial filter file (heuristics can be modified)
- `scorep-score -f filter`: testing the effects of a proposed filter file

## Goals of scoring and filtering

---

- Evaluate how *expensive* measurement of various regions is
  - Time cost is roughly fixed per event
  - Short functions are relatively more expensive
  - Space cost for tracing is linear in number of events
- Determine which expensive regions are *unnecessary* to measure
  - Frequently called, short execution, and non-scaling behavior
- Repeat the measurement, with those regions *filtered* at runtime to reduce overhead
  - Reduce space cost to zero and time cost to a hash comparison and a couple of branches
- (Optional) Apply the filter at *compile-time* in order to further reduce overhead
  - Eliminates the hash and branches; often not needed

# NPB-MZ-MPI / BT summary analysis result scoring

```
% scorep-score scorep-btmz-D-32x7/profile.cubex
```

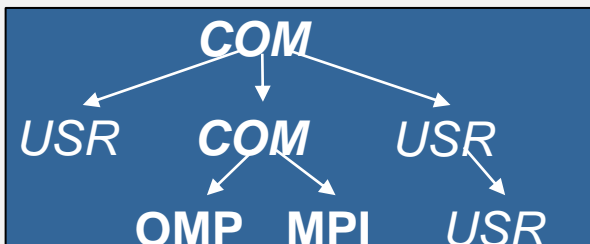
```
Estimated aggregate size of event trace:          3332GB
Estimated requirements for largest trace buffer (max_buf): 105GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 105GB
(warning: The memory requirements cannot be satisfied by Score-P to avoid
intermediate flushes when tracing. Set SCOREP_TOTAL_MEMORY=4G to get the
maximum supported memory or reduce requirements using USR regions filters.)
```

3332 GB total memory  
105 GB per rank!

flt	type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
	ALL	112,035,087,899	137,518,952,689	29382.90	100.0	0.21	ALL
	USR	111,908,772,534	137,419,580,865	11380.33	38.7	0.08	USR
	OMP	121,893,856	95,047,680	17750.19	60.4	186.75	OMP
	COM	2,935,270	3,612,640	14.97	0.1	4.14	COM
	MPI	1,588,606	711,472	237.42	0.8	333.70	MPI
	SCOREP	41	32	0.00	0.0	56.03	SCOREP

## Region/callpath classification

- **MPI** pure MPI functions
- **OMP** pure OpenMP regions
- **USR** user-level computation
- **COM** "combined" USR+OpenMP/MPI
- **SCOREP** measurement internals
- **ANY/ALL** aggregate of all region types



# NPB-MZ-MPI / BT summary analysis report breakdown

```
% scorep-score -r scorep-btmz-D-32x7/profile.cubex
```

```
[...]
```

```
[...]
```

flt	type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
	ALL	112,035,087,899	137,518,952,689	29382.90	100.0	0.21	ALL
	USR	111,908,772,534	137,419,580,865	11380.33	38.7	0.08	USR
	OMP	121,893,856	95,047,680	17750.19	60.4	186.75	OMP
	COM	2,935,270	3,612,640	14.97	0.1	4.14	COM
	MPI	1,588,606	711,472	237.42	0.8	333.70	MPI
	SCOREP	41	32	0.00	0.0	56.03	SCOREP
	USR	36,395,971,872	44,677,967,872	5074.31	17.3	0.11	binvrhs
	USR	36,395,971,872	44,677,967,872	3479.03	11.8	0.08	matmul_sub
	USR	36,395,971,872	44,677,967,872	2510.98	8.5	0.06	matvec_sub
	USR	957,566,506	1,152,495,616	152.76	0.5	0.13	lhsinit
	USR	957,566,506	1,152,495,616	103.12	0.4	0.09	binvrhs
	USR	878,133,152	1,080,078,336	60.03	0.2	0.06	exact_solution
	OMP	9,782,976	3,598,336	0.53	0.0	0.15	!\$omp parallel @exch_qbc.f:204
	OMP	9,782,976	3,598,336	0.52	0.0	0.14	!\$omp parallel @exch_qbc.f:215
	OMP	9,782,976	3,598,336	0.49	0.0	0.14	!\$omp parallel @exch_qbc.f:244
	OMP	9,782,976	3,598,336	0.55	0.0	0.15	!\$omp parallel @exch_qbc.f:255

```
[...]
```

Majority of the  
105 GB just for these 6  
regions

## NPB-MZ-MPI / BT summary analysis report filtering

---

```
% scorep-score -g scorep-btmz-D-32x7/profile.cubex
```

An initial filter file template has been generated:

```
'initial_scorep.filter'
```

To use this file for filtering at run-time, set the respective Score-P variable:

```
SCOREP_FILTERING_FILE=initial_scorep.filter
```

For compile-time filtering 'scorep' has to be provided with the '--instrument-filter' option:

```
$ scorep --instrument-filter=initial_scorep.filter
```

Compile-time filtering depends on support in the used Score-P installation.

The filter file is annotated with comments, please check if the selection is suitable for your purposes and add or remove functions if needed.

- Report scoring with prospective filter listing 3 USR regions

# NPB-MZ-MPI / BT summary analysis report filtering

```

% scorep-score -r -f initial_scorep.filter \
> scorep-btmz-D-32x7/profile.cubex
flt      type      max_buf[B]      visits  time[s]  time[%]  time/visit[us]  region
-        ALL  112,035,087,899  137,518,952,689  29382.90  100.0    0.21            ALL
-        USR  111,908,772,534  137,419,580,865  11380.33   38.7    0.08            USR
-        OMP   121,893,856     95,047,680     17750.19   60.4    186.75          OMP
-        COM    2,935,270      3,612,640      14.97     0.1     4.14            COM
-        MPI    1,588,606      711,472       237.42    0.8    333.70          MPI
-        SCOREP  41             32             0.00     0.0    56.03          SCOREP

*        ALL  2,919,186,417   3,485,049,073  18318.59   62.3    5.26            ALL-FLT
+        FLT  109,187,915,616  134,033,903,616  11064.32   37.7    0.08            FLT
*        USR  2,792,819,848   3,385,677,249   316.01     1.1     0.09            USR-FLT
-        OMP   121,893,856     95,047,680     17750.19   60.4    186.75          OMP-FLT
*        COM    2,935,270      3,612,640      14.97     0.1     4.14            COM-FLT
-        MPI    1,588,606      711,472       237.42    0.8    333.70          MPI-FLT
-        SCOREP  41             32             0.00     0.0    56.03          SCOREP-FLT

+        USR  36,395,971,872  44,677,967,872  5074.31    17.3    0.11            binvrhs
+        USR  36,395,971,872  44,677,967,872  3479.03    11.8    0.08            matmul_sub
+        USR  36,395,971,872  44,677,967,872  2510.98     8.5    0.06            matvec_sub
-        USR  957,566,506  1,152,495,616  152.76     0.5    0.13            lhsinit
-        USR  957,566,506  1,152,495,616  103.12     0.4    0.09            binvrhs
-        USR  878,133,152  1,080,078,336   60.03     0.2    0.06            exact_solution

```

Filtered routines  
marked with '+'

- Score report breakdown by region
- Adapt initial filter file to include the 3 additional routines

# NPB-MZ-MPI / BT summary analysis report filtering

```
% scorep-score -r -f initial_scorep.filter \
> scorep-btmz-D-32x7/profile.cubex
Estimated aggregate size of event trace: 3871MB
Estimated requirements for largest trace buffer (max_buf): 122MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 136MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=136MB to avoid intermediate flushes
or reduce requirements using USR regions filters.)
[...]
```

+	USR	36,395,971,872	44,677,967,872	5063.46	17.2	0.11	binvrhs
+	USR	36,395,971,872	44,677,967,872	3469.07	11.8	0.08	matmul_sub
+	USR	36,395,971,872	44,677,967,872	2493.53	8.5	0.06	matvec_sub
+	<b>USR</b>	<b>957,566,506</b>	<b>1,152,495,616</b>	<b>152.54</b>	<b>0.5</b>	<b>0.13</b>	<b>lhsinit</b>
+	<b>USR</b>	<b>957,566,506</b>	<b>1,152,495,616</b>	<b>102.90</b>	<b>0.4</b>	<b>0.09</b>	<b>binvrhs</b>
+	<b>USR</b>	<b>878,133,152</b>	<b>1,080,078,336</b>	<b>59.91</b>	<b>0.2</b>	<b>0.06</b>	<b>exact_solution</b>

```
% cat initial_scorep.filter
SCOREP_REGION_NAMES_BEGIN
EXCLUDE
lhsinit
binvrhs
exact_solution

# type=USR max_buf= 36,395,971,872 visits= 44,677,967,872, ...
# name='binvrhs'
# file='BT-MZ/solve_subs.f'
MANGLED binvrhs_
```

3.9 GB of memory  
in total,  
136 MB per rank!

Manually added entries,  
Space separated and bash-  
like wildcards \*, ?, []

- Refined filter reduces the requirements to useful sizes.
- More information on manual and automatic filtering in the user documentation and `scorep-score --help`

## Adding PAPI Performance counters

- Estimating the memory requirements due to recorded counters:

```
% <editor> scorep.slurm
...
#SBATCH -J bt-scorep-filter
% scorep-score -f initial_scorep.filter -c 2 scorep-btmz-D-32x7/profile.cubex
```

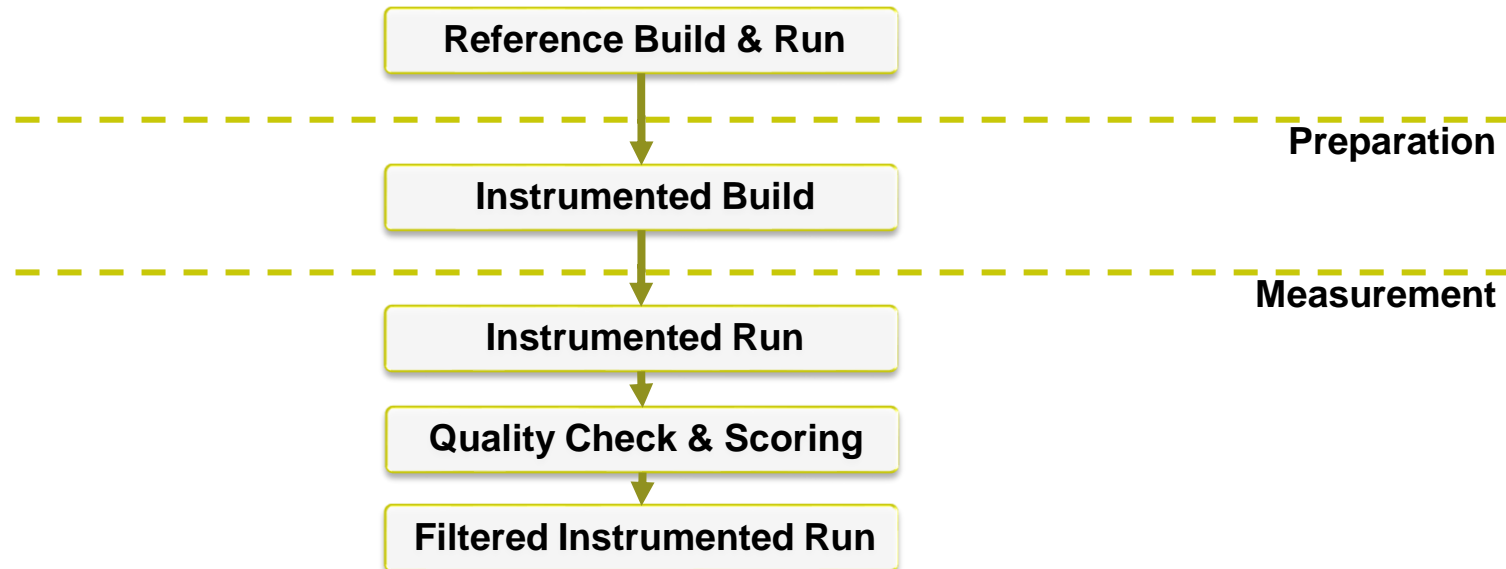
```
Estimated aggregate size of event trace: 10GB
Estimated requirements for largest trace buffer (max_buf): 297MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 311MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=311MB to avoid intermediate tables
or reduce requirements using USR regions filters.)
```

Increased memory requirements from the prior 136 MB per rank!

- With a PAPI module loaded: `papi_avail` for a list of available counters

```
export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
```

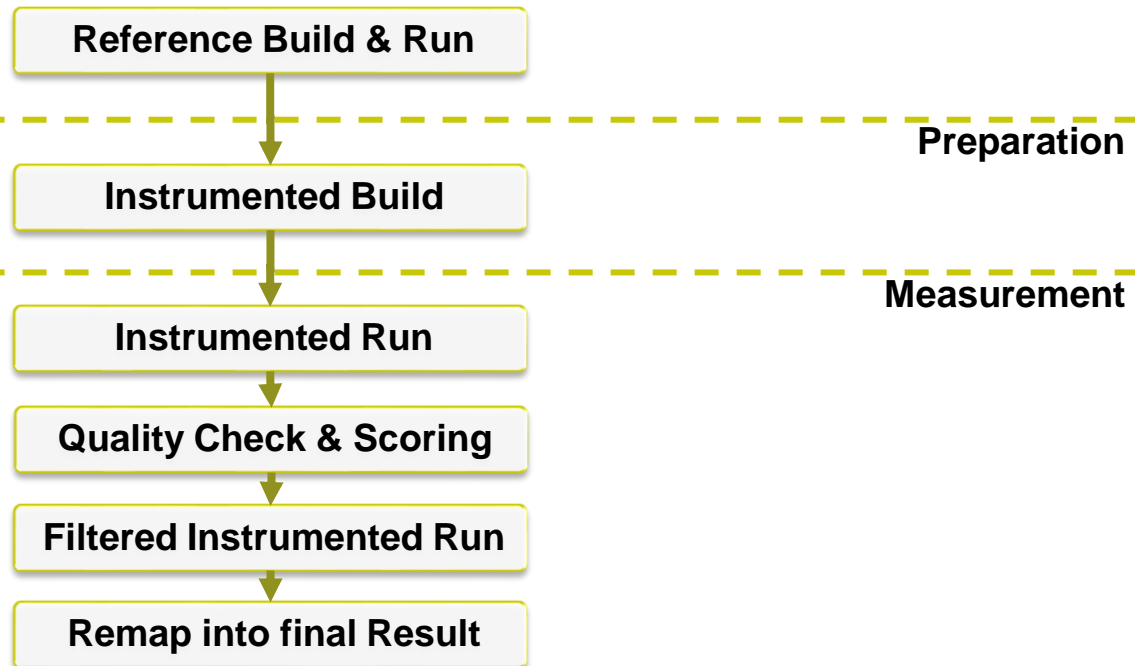
# Performance analysis workflow: Filtered Run



## Filtered Instrumented Run

- Using the generated and modified filter file via `SCOREP_FILTERING_FILE=name`
- If necessary modify and rerun, but ideally that should work via the scoring preview
- The result should have a `profile.cubex` that is technically usable

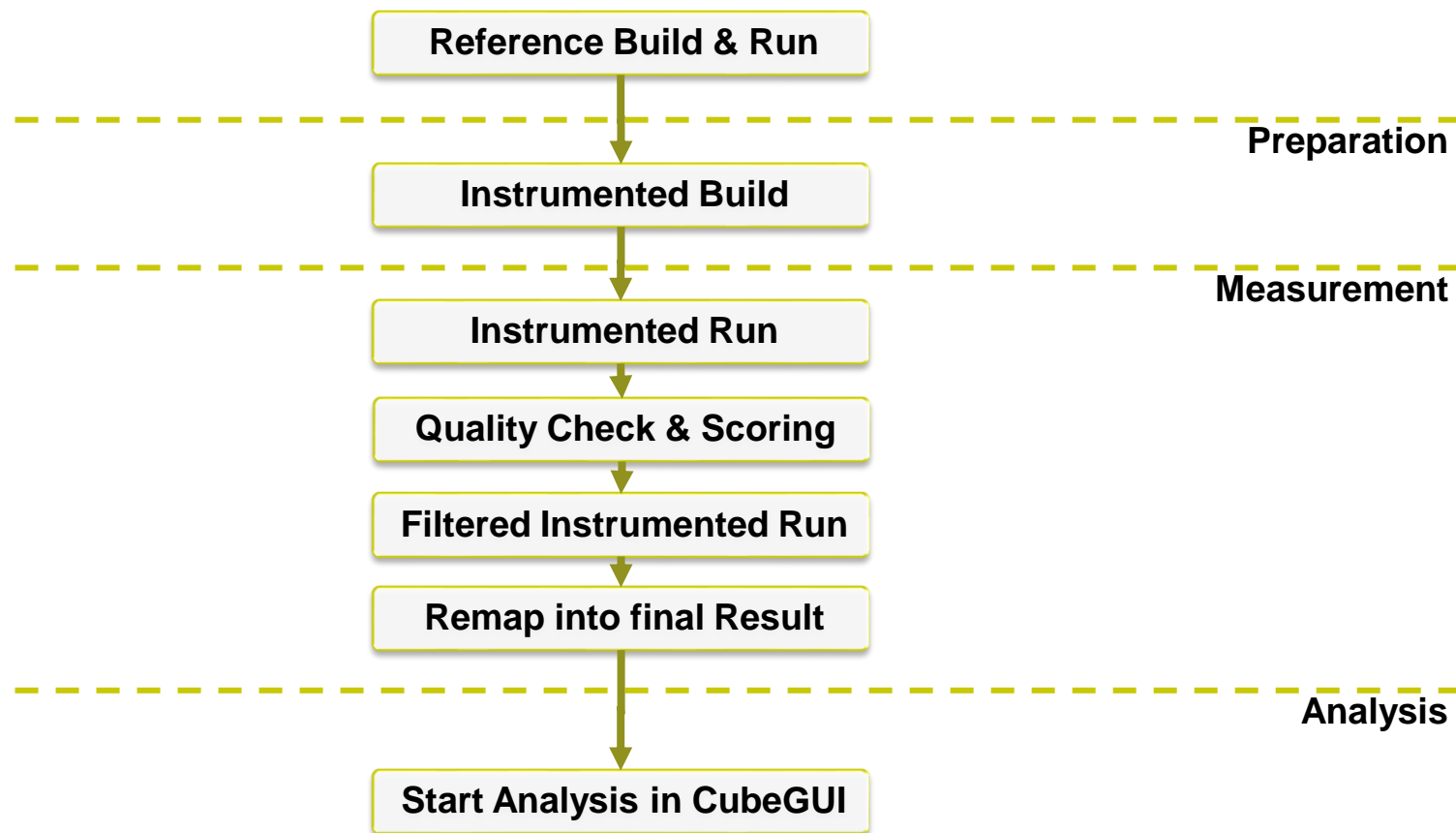
# Performance analysis workflow: Postprocessing



## Remap into final Result

- Postprocessing step that generates additional metrics and data
- Results in a measurement ready for analysis
- The remap step can be done by `cube_remap2 -s -d profile.cubex`
- `-s`: adds additional metrics
- `-d`: precalculates values (tradeoff size vs. calculation effort while analysing)

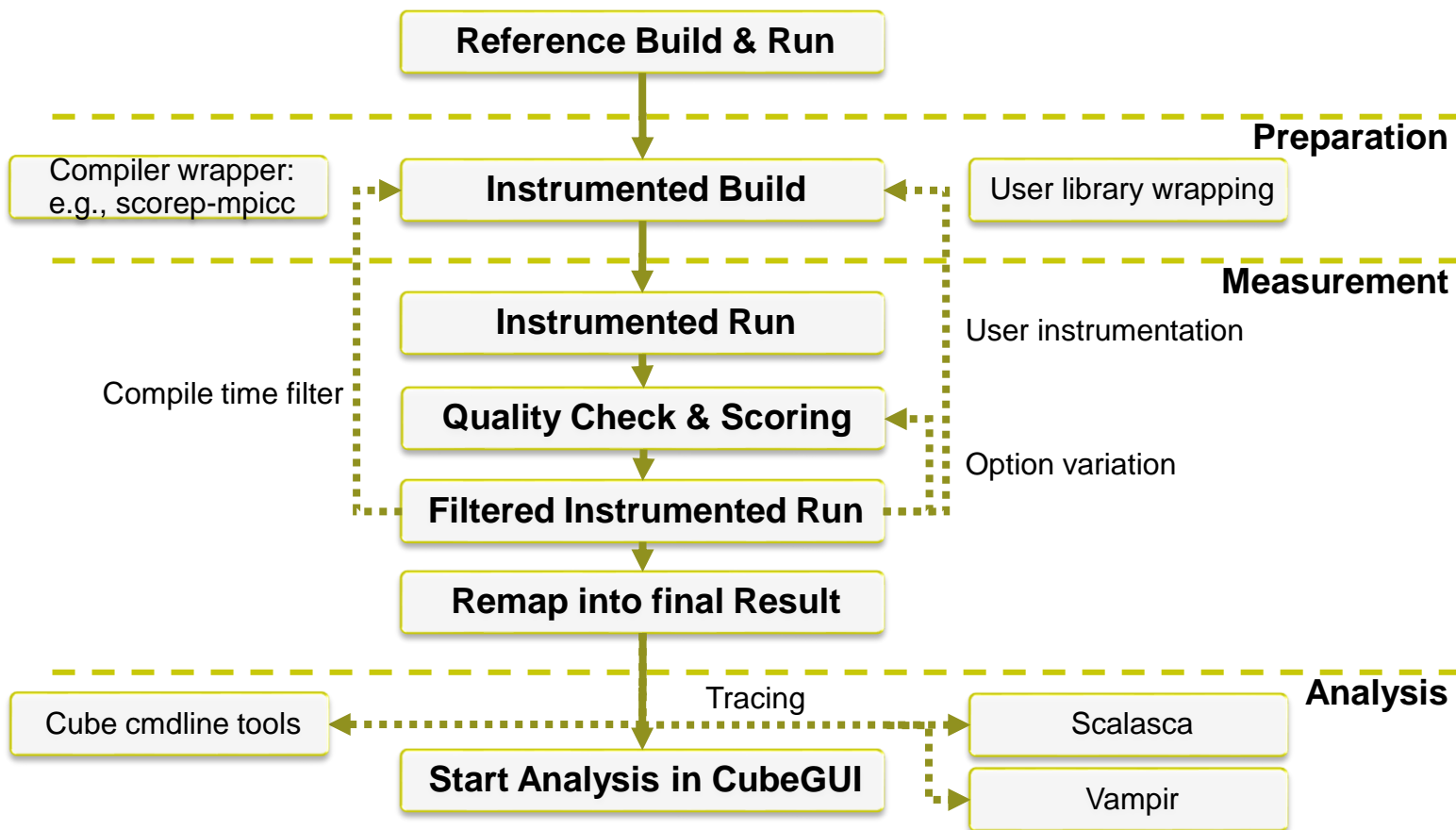
# Performance analysis workflow: Start Analysis



## Start Analysis in CubeGUI

- Open the result in the CubeGUI
- The GUI is available as:
  - To build from source
  - Various prebuilt executables
- The cubex files are transferable and the only element required to run
- For source code references the application sources need to be accessible

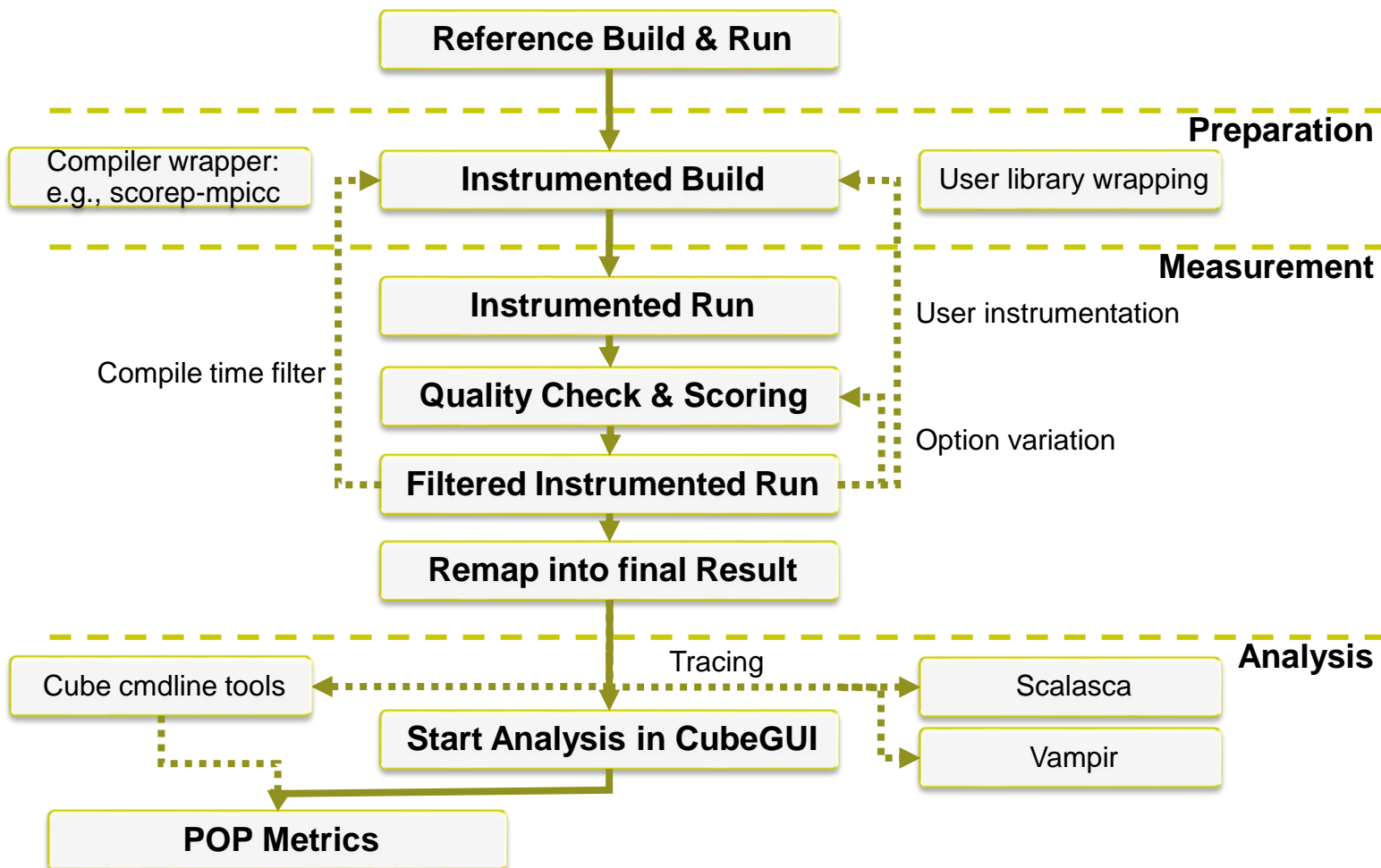
# Performance analysis workflow: Outlook



## Foreshadowing

- The straightforward linear graph can become more complicated, depending on software environment, application requirements, measured paradigms and various other settings
- Going into tracing as next step will be covered tomorrow
- See reference slides at the end

# Performance analysis workflow: POP Metrics



## POP Metrics

- Additional Requirements:
  - Hardware Counters
  - (Optional Metrics from Tracing)
    - => Scalasca
- Calculation via GUI or cmd line `cube_pop_metrics`
- More complex selections easier in GUI

## Calculating POP metrics on the cmd line: cube\_pop\_metrics

### % cube\_pop\_metrics -h

Usage: cube\_pop\_metrics [-i] [-a <pop analysis>] [-c <FOI(s)>] [-h] [-? <pop analysis>] <cube1> <cube2> ..

- a Name of the POP analysis. Options: mpi, hybrid-add, hybrid-mult, **bsc**. Default: mpi
- c Name(s) or ID(s) of FOI(s). Default: calltree root(s). Comma separated
- i Performs the calculation individually for every listed FOI.  
Default: if omitted, the calculation is performed in an accumulated manner
- h Help; Output a brief help message.
- ? Help; Output a long detailed help message.

```
POP Metric                                Profile 0
-----
Hybrid Parallel Efficiency                0.770967
 * Hybrid Load Balance Efficiency         0.851013
 * Hybrid Communication Efficiency        0.905941
MPI Parallel Efficiency                   0.896192
 * MPI Load Balance                       0.949225
 * MPI Communication Efficiency           0.944130
 * * * MPI Serialisation Efficiency       nan
 * * * MPI Transfer Efficiency           nan
OpenMP Parallel Efficiency                 0.860270
 * OpenMP Load Balance Efficiency         0.896534
 * OpenMP Communication Efficiency        0.959551
...
```

Some values not available in  
pure profile measurements  
=> Provided by Scalasca  
through tracing

## Score-P: Further information

---

- Scalable Performance Measurement Infrastructure for Parallel Codes
  - Instrumenter, libraries, and tools to generate profile and trace measurements
  - Bundled with OTF2 (tracing), OPARI2 (OpenMP instrumentation), CubeWriter, and CubeLib (profiling)
- Available under 3-clause BSD open-source license
- Documentation & sources:
  - <https://score-p.org>
- User guide also part of installation:
  - `<prefix>/share/doc/scorep/pdf/scorep.pdf`
- Contact:
  - mailto: [support@score-p.org](mailto:support@score-p.org)

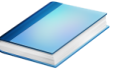


## Reference Material: Specialized Measurements and Analysis Options

---



# Mastering build systems



- Hooking up the Score-P instrumenter `scorep` into complex build environments like *Autotools* or *CMake* was always challenging
- Score-P provides convenience wrapper scripts to simplify this
- *Autotools* and *CMake* need the used compiler already in the *configure step*, but instrumentation should not happen in this step, only in the *build step*

```
% SCOREP_WRAPPER=off \  
> cmake .. \  
> -DCMAKE_C_COMPILER=scorep-icc \  
> -DCMAKE_CXX_COMPILER=scorep-icpc
```

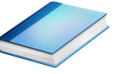
Disable instrumentation in the *configure step*

Specify the wrapper scripts as the compiler to use

- Allows to pass additional options to the Score-P instrumenter and the compiler via environment variables without modifying the *Makefiles*
- Run `scorep-wrapper --help` for a detailed description and the available wrapper scripts of the Score-P installation

# User instrumentation

---



- No replacement for automatic compiler instrumentation
- Can be used to further subdivide functions
  - E.g., multiple loops inside a function
- Can be used to partition application into coarse grain phases
  - E.g., initialization, solver, & finalization
- Enabled with `--user` flag to Score-P instrumenter
- Available for Fortran / C / C++

# User instrumentation: Fortran



```
#include <scorep/SCOREP_User.inc>

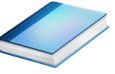
subroutine foo(...)
  ! Declarations
  SCOREP_USER_REGION_DEFINE( solve )

  ! Some code...
  SCOREP_USER_REGION_BEGIN( solve, "<solver>", \
                           SCOREP_USER_REGION_TYPE_LOOP )

  do i=1,100
    [...]
  end do
  SCOREP_USER_REGION_END( solve )
  ! Some more code...
end subroutine
```

- Requires processing by the C preprocessor
  - For most compilers, this can be automatically achieved by having an uppercase file extension, e.g., main.F or main.F90

# User instrumentation: C/C++

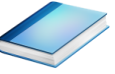


```
#include <scorep/SCOREP_User.h>

void foo()
{
    /* Declarations */
    SCOREP_USER_REGION_DEFINE( solve )

    /* Some code... */
    SCOREP_USER_REGION_BEGIN( solve, "<solver>",
                             SCOREP_USER_REGION_TYPE_LOOP )
    for (i = 0; i < 100; i++)
    {
        [...]
    }
    SCOREP_USER_REGION_END( solve )
    /* Some more code... */
}
```

# User instrumentation: C++



```
#include <scorep/SCOREP_User.h>

void foo()
{
    // Declarations

    // Some code...
    {
        SCOREP_USER_REGION( "<solver>",
                           SCOREP_USER_REGION_TYPE_LOOP )
        for (i = 0; i < 100; i++)
        {
            [...]
        }
    }
    // Some more code...
}
```

# Score-P measurement control API



- Can be used to temporarily disable measurement for certain intervals
  - Annotation macros ignored by default
  - Enabled with `--user` flag

```
#include <scorep/SCOREP_User.inc>

subroutine foo(...)
  ! Some code...
  SCOREP_RECORDING_OFF()
  ! Loop will not be measured
  do i=1,100
    [...]
  end do
  SCOREP_RECORDING_ON()
  ! Some more code...
end subroutine
```

Fortran (requires C preprocessor)

```
#include <scorep/SCOREP_User.h>

void foo(...) {
  /* Some code... */
  SCOREP_RECORDING_OFF()
  /* Loop will not be measured */
  for (i = 0; i < 100; i++) {
    [...]
  }
  SCOREP_RECORDING_ON()
  /* Some more code... */
}
```

C / C++

# Enriching measurements with performance counters



- Record metrics from PAPI:

```
% export SCOREP_METRIC_PAPI=PAPI_TOT_CYC
% export SCOREP_METRIC_PAPI_PER_PROCESS=PAPI_L3_TCM
```

- Use PAPI tools to get available metrics and valid combinations:

```
% papi_avail
% papi_native_avail
```

- Record metrics from Linux perf:

```
% export SCOREP_METRIC_PERF=cpu-cycles
% export SCOREP_METRIC_PERF_PER_PROCESS=LLC-load-misses
```

- Use the `perf` tool to get available metrics and valid combinations:

```
% perf list
```

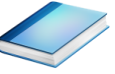
- Write your own metric plugin

- Repository of available plugins: <https://github.com/score-p>

Only the master thread records the metric (assuming all threads of the process access the same L3 cache)

# Mastering application memory usage

---

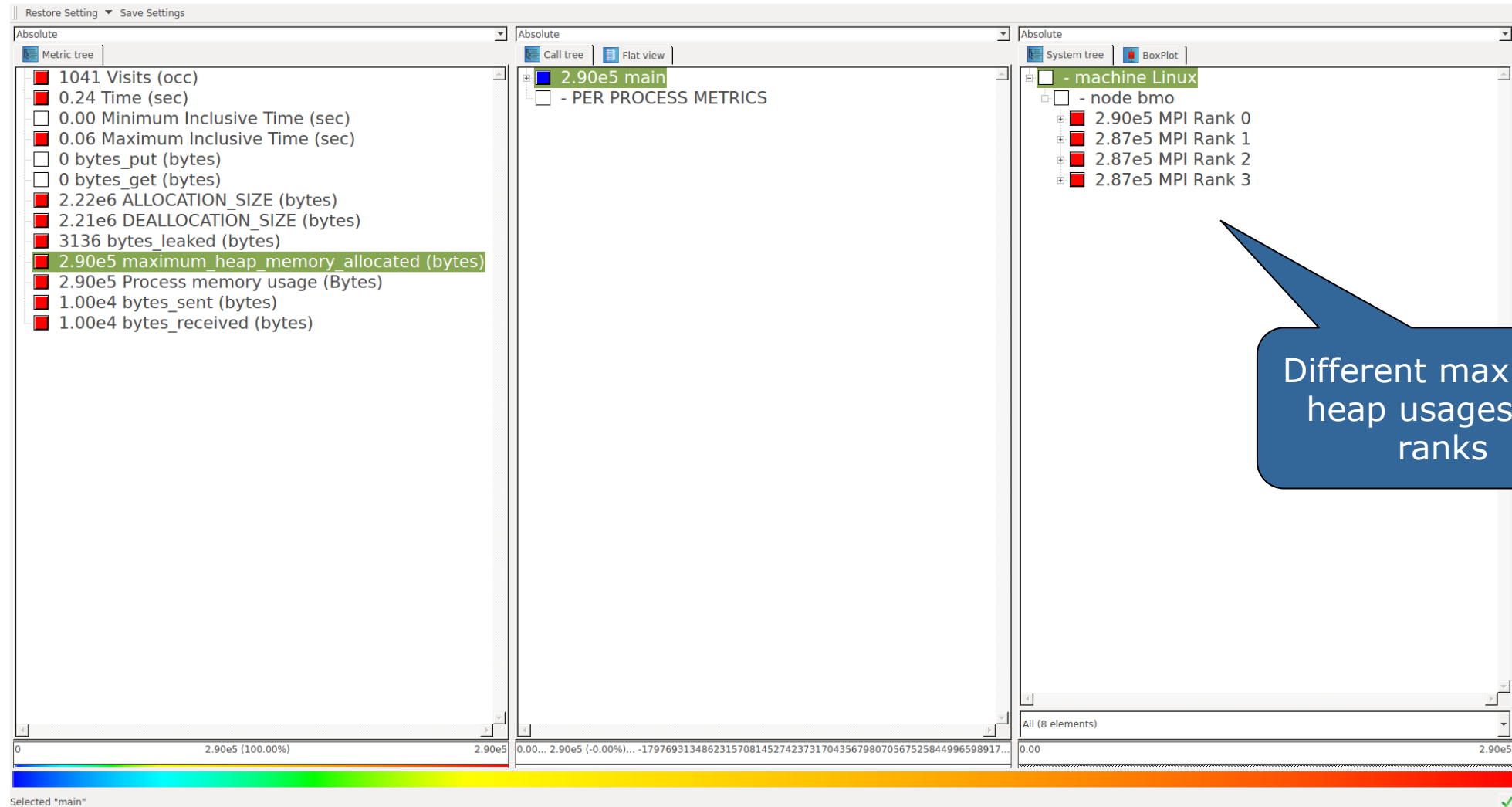
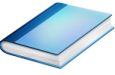


- Determine the maximum heap usage per process
- Find high frequent small allocation patterns
- Find memory leaks
- Support for:
  - C, C++, MPI, and SHMEM
  - Profile and trace generation (profile recommended)
    - Memory leaks are recorded only in the profile
    - Resulting traces are not supported by Scalasca yet

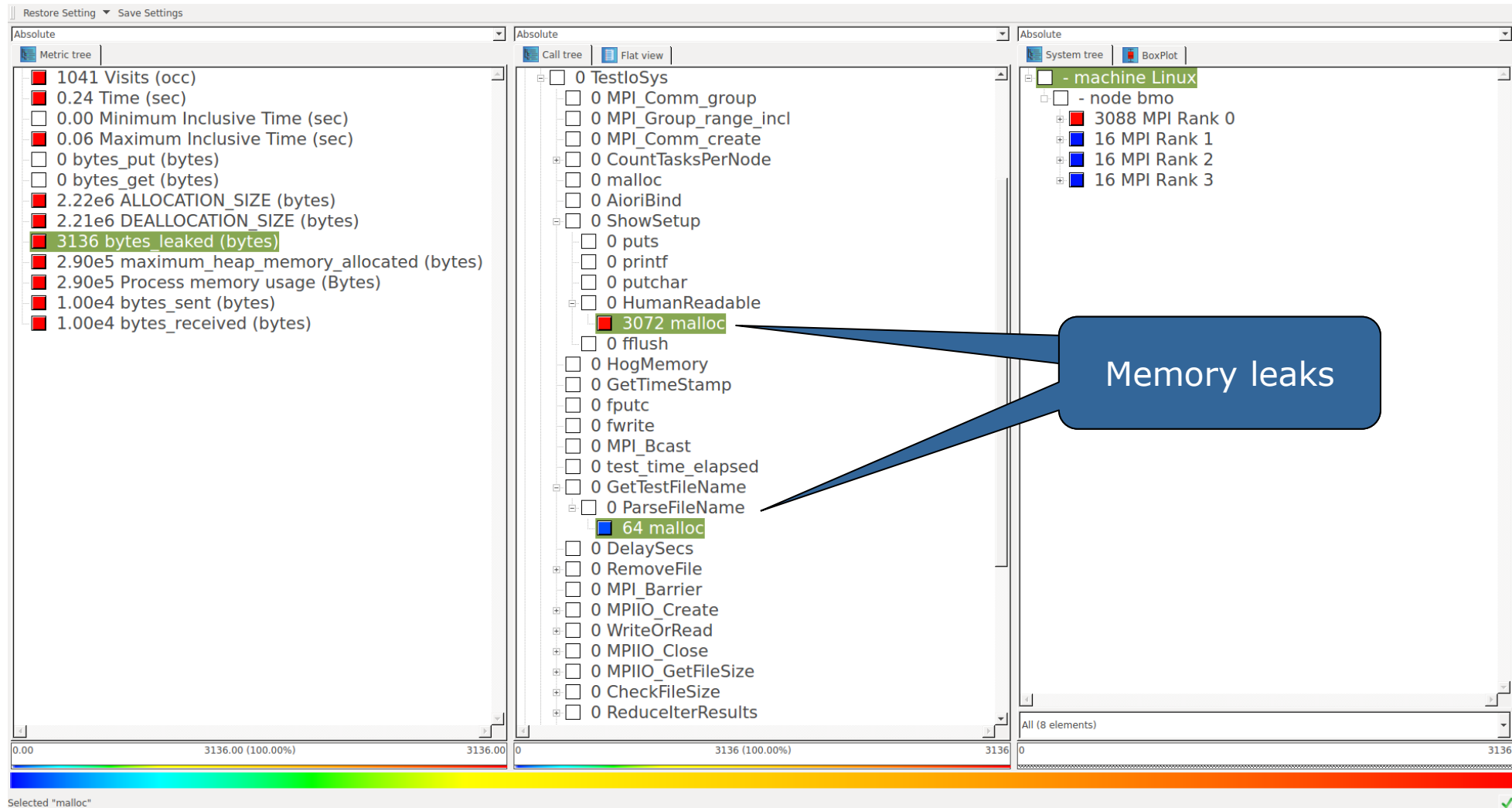
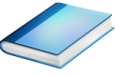
```
% export SCOREP_MEMORY_RECORDING=true  
% export SCOREP_MPI_MEMORY_RECORDING=true
```

- Set new configuration variable to enable memory recording

# Mastering application memory usage



# Mastering application memory usage



# Mastering accelerators

---



- Record CUDA applications and device activities

```
% export SCOREP_CUDA_ENABLE=gpu, kernel, idle
```

- Record HIP applications and device activities

```
% export SCOREP_HIP_ENABLE=all
```

- Record OpenCL applications and device activities

```
% export SCOREP_OPENCL_ENABLE=api, kernel
```

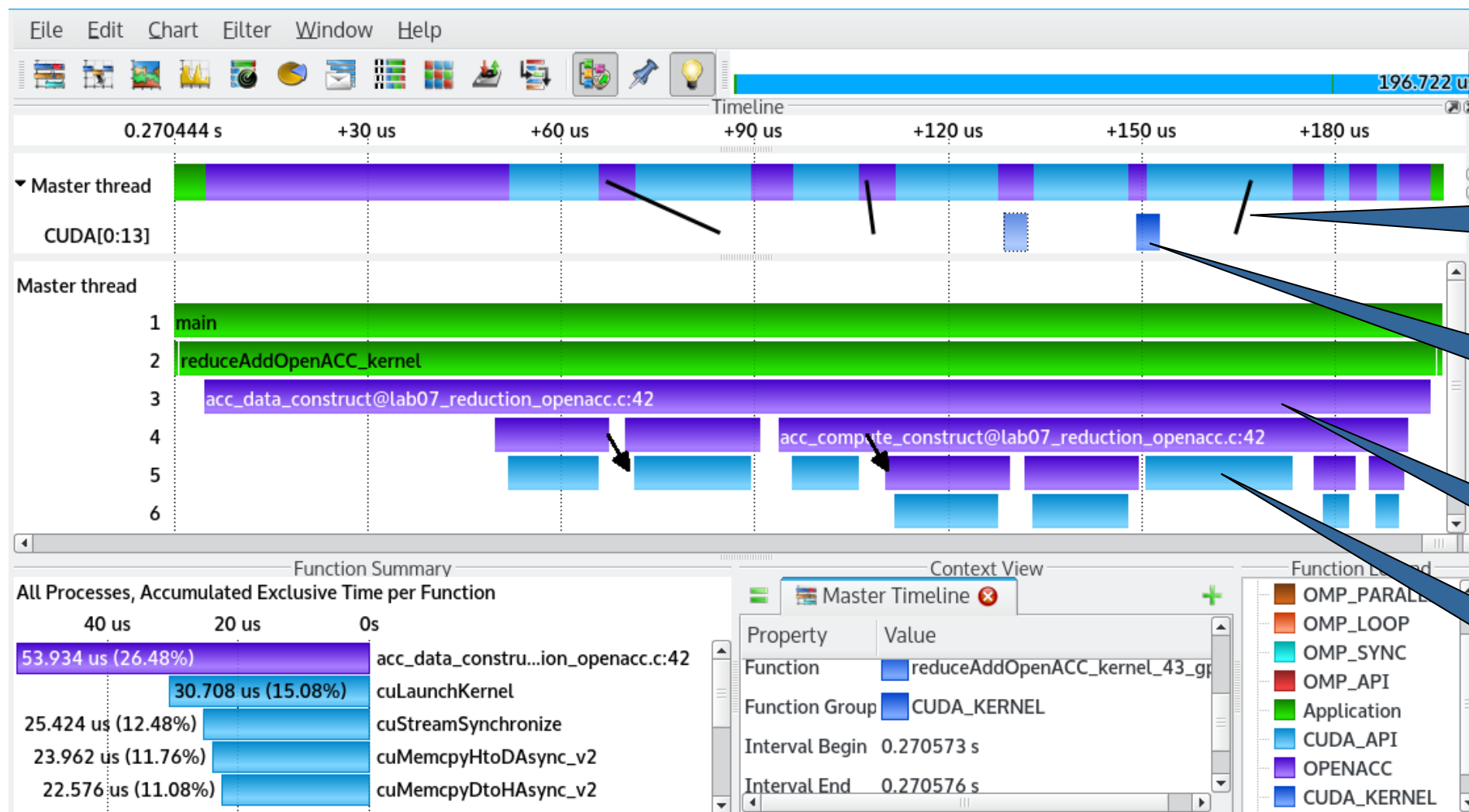
- Record OpenACC applications

```
% export SCOREP_OPENACC_ENABLE=yes
```

- Can be combined with CUDA if it is a NVIDIA device

```
% export SCOREP_CUDA_ENABLE=kernel
```

# Mastering accelerators



Host-Device memory transfers

Device activities

OpenACC directives

CUDA API calls

# Hybrid measurement with sampling

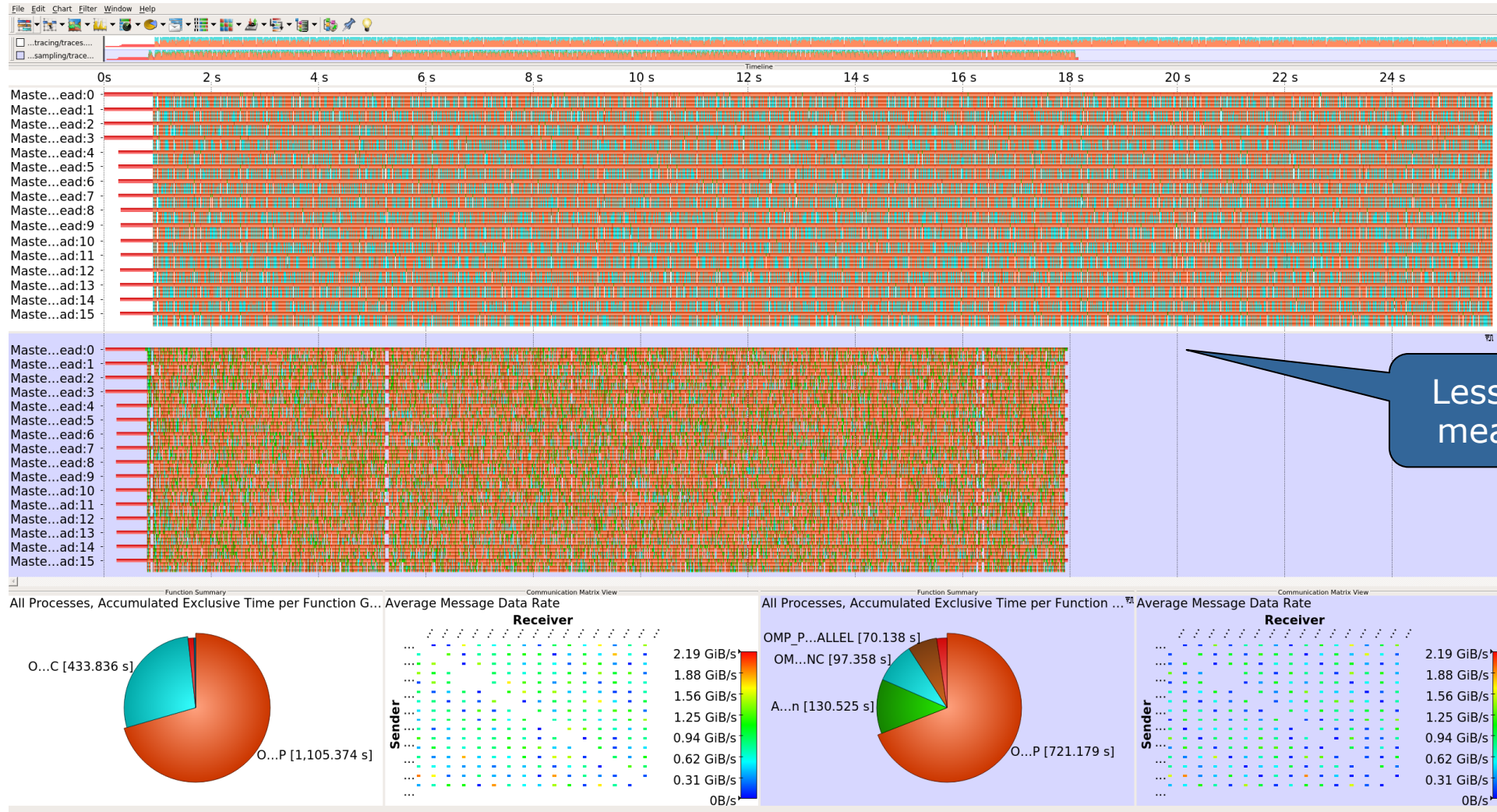
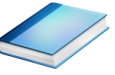


- Automatic compiler instrumentation greatly disturbs C++ applications because of frequent/short function calls ⇒ Use sampling instead
- Novel combination of sampling events and instrumentation of MPI, OpenMP, ...
  - Sampling replaces only compiler instrumentation (use `--nocompiler`)
  - Instrumentation is still used for parallel activities (MPI, OpenMP, CUDA, I/O)
- Supports profile and trace generation

```
% export SCOREP_ENABLE_UNWINDING=true  
% # use the default sampling frequency  
% #export SCOREP_SAMPLING_EVENTS=perf_cycles@2000000
```

- Set new configuration variable to enable sampling

# Mastering C++ applications



Less disturbed measurement

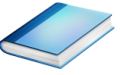
# Instrumenting calls to 3<sup>rd</sup> party libraries

---



- Score-P does not record function calls to non-instrumented external libraries
- Increase insight into the behavior of the application
  - How does the application use the external library?
  - How does this compares to the usage of other libraries?
- Manual user instrumentation of the application using the library should be avoided
- Vendor provided libraries cannot be instrumented, but API is provided in headers

# Instrumenting calls to 3<sup>rd</sup> party libraries: Library wrapper generator



- Workflow to generate library wrappers for most C/C++ libraries
- Tailored towards users of the external library, not users of Score-P
- Results can be shared with multiple users
- Workflow driver `scorep-libwrap-init --help` provides instructions

Only once

## Library wrapper creator

1. Initialize the working directory
2. Add library headers
3. Create a simple example application
4. Further configure the build parameters
5. Build the wrapper
6. Verify the wrapper
7. Install the wrapper
8. Verify the installed wrapper

## Library wrapper user

9. Use the wrapper

Often

# Instrumenting calls to 3<sup>rd</sup> party libraries: Workflow



- Start workflow by telling `scorep-libwrap-init` how you would compile and link an application, e.g., using FFTW

```
% scorep-libwrap-init \
> --name=fftw \
> --prefix=$PREFIX \
> -x c \
> --cppflags="-O3 -DNDEBUG -openmp -I$FFTW_INC" \
> --ldflags="-L$FFTW_LIB" \
> --libs="-lfftw3f -lfftw3" \
> working_directory
```

Omit to install into Score-P

Flags used to compile/link

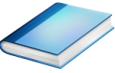
Working directory can be archived for later rebuild

- Generate and build wrapper

```
% cd working_directory
% ls # (Check README.md for instructions)
% make # Generate and build wrapper
% make check # See if header analysis matches symbols
% make install #
% make installcheck # More checks: Linking etc.
```

Tells you how to use the wrapper with Score-P

# Instrumenting calls to 3<sup>rd</sup> party libraries: Usage and result



- List of available wrappers:

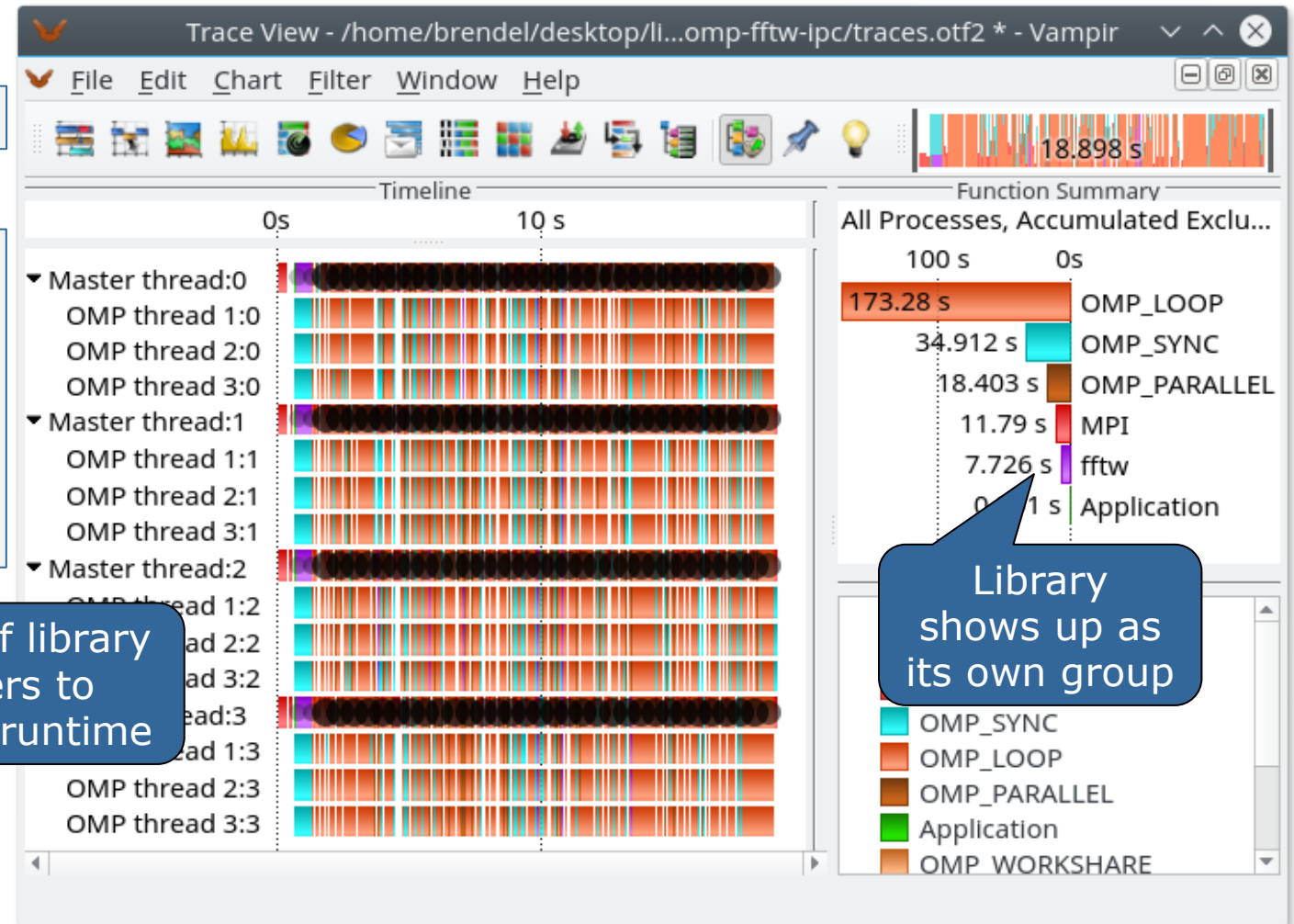
```
% scorep-info libwrap-summary
```

- Instrumentation:

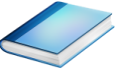
```
# instrument application as usual
% cd <application>
% make clean
% make
% export \
SCOREP_LIBWRAP_ENABLE=fftw
# run application as usual
```

- MPI + OpenMP
- Calls to FFTW library

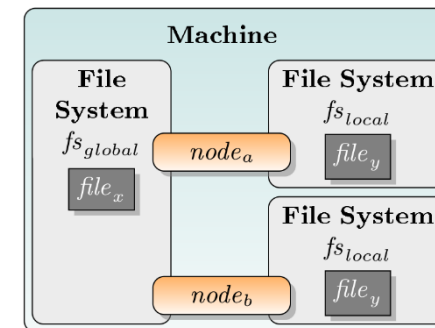
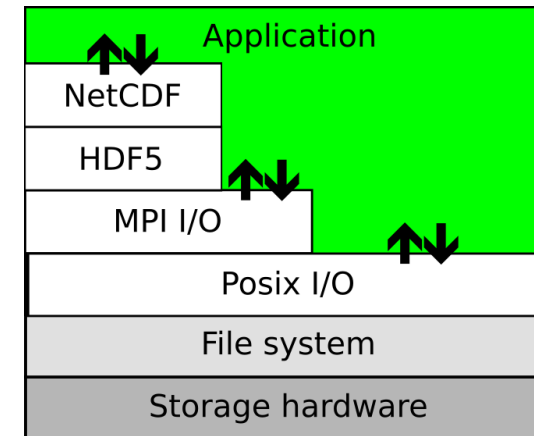
Pass list of library wrappers to enable at runtime



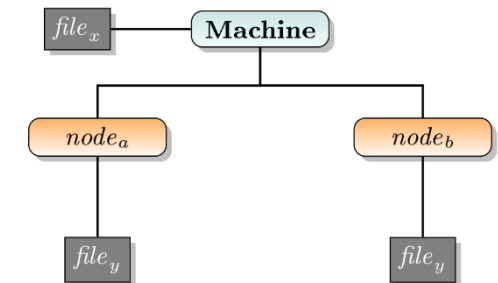
# Mastering File I/O



- Omnipresent in today's HPC applications
- Record interaction between multiple layers
  - MPI I/O (`MPI_File_open`)
  - ISO C I/O (`fopen`)
  - POSIX I/O (`open`, interface to OS)
- System tree information determine whether file resides in a shared file system
- High level of detail
  - => Trace data might increase dramatically



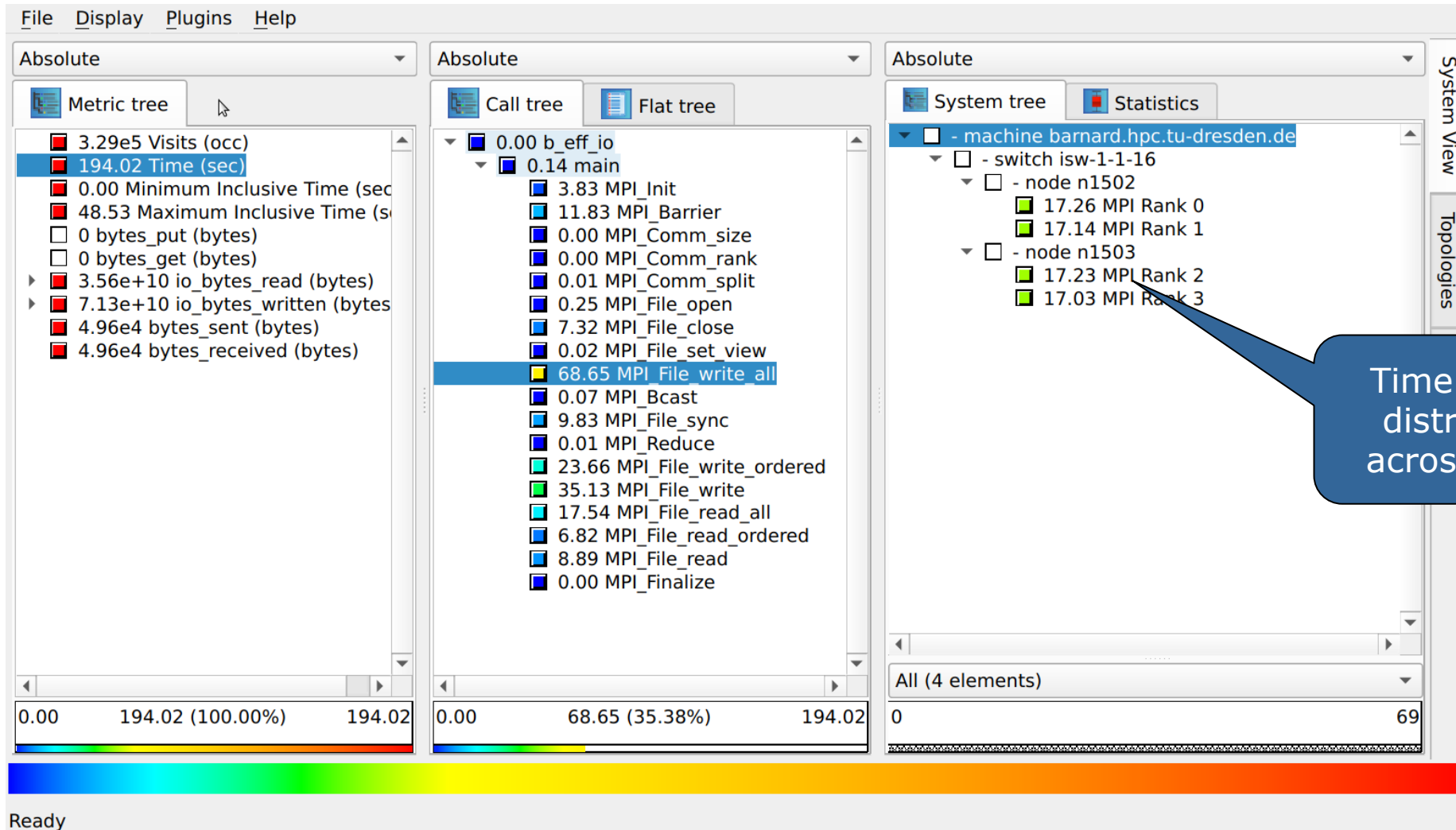
(a) Hardware topology



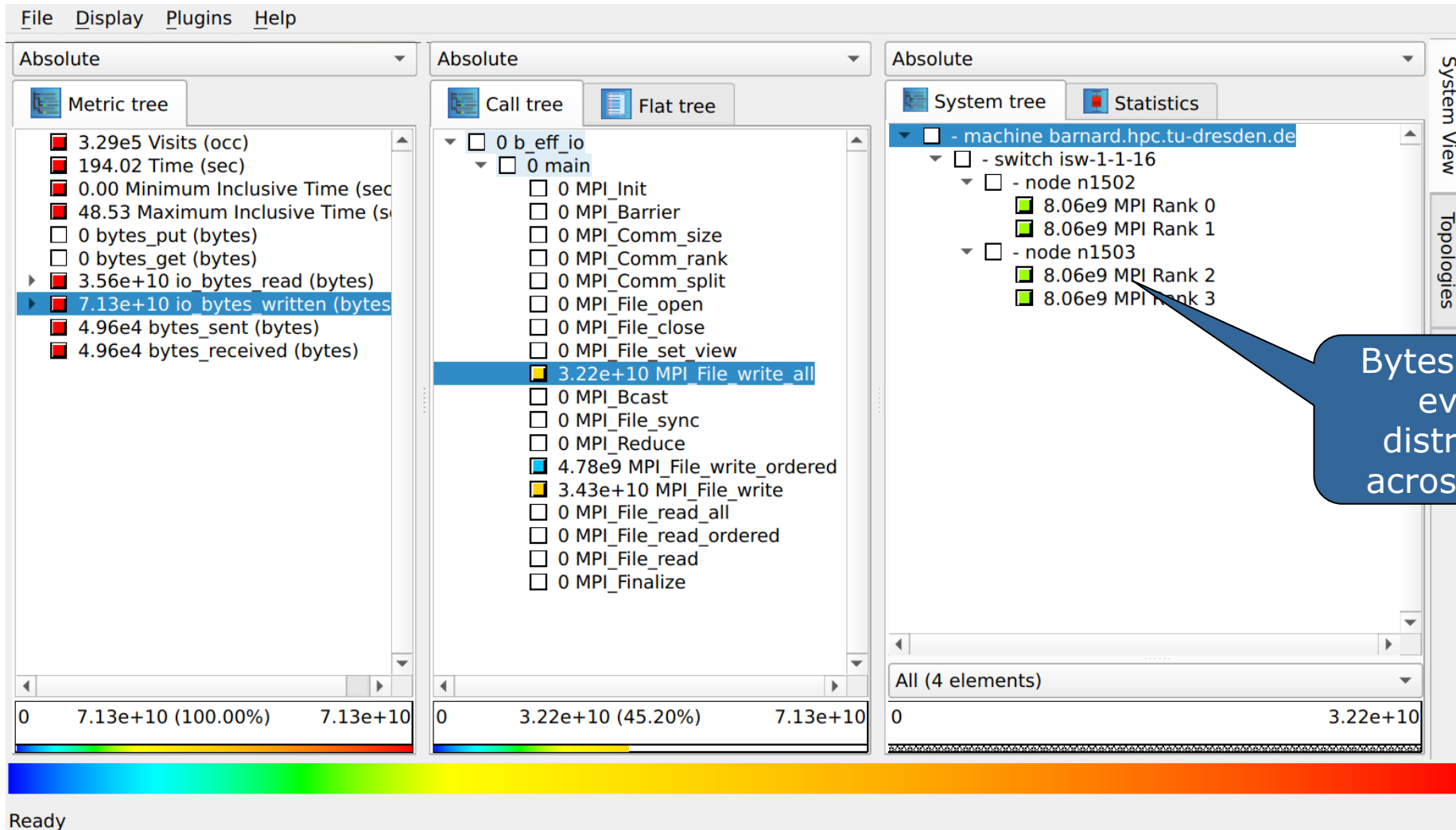
(b) System tree representation

NetCDF & HDF5 will be supported later

# Mastering File I/O: Result visualization



# Mastering File I/O: Result visualization



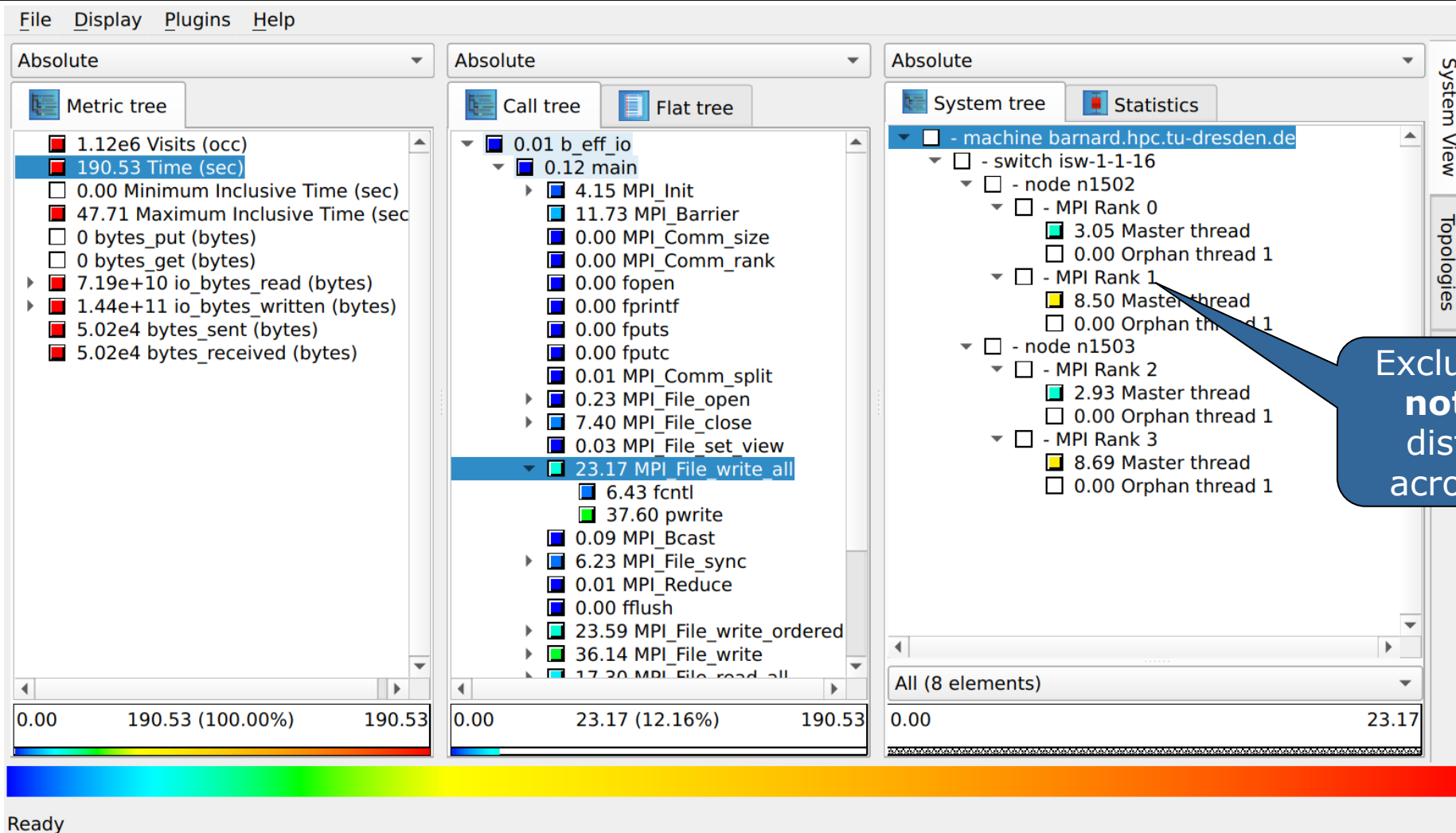
# Mastering File I/O: B\_EFF I/O hands-on

---

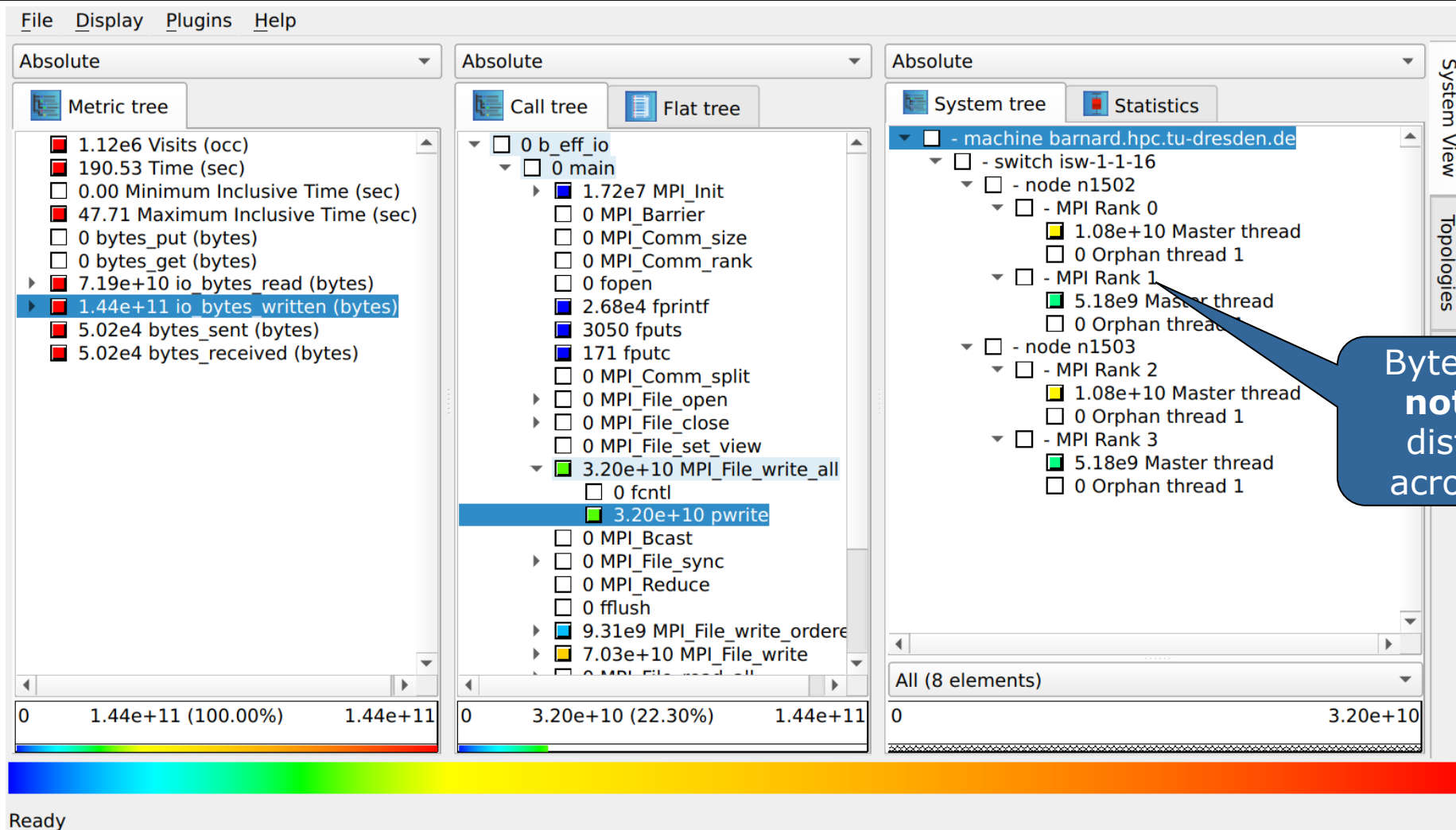
- Enabling ISO C and POSIX I/O instrumentation
- Instrumentation does require threading support

```
% scorep-mpiicc -g -O3 -o b_eff_io b_eff_io.c
% export SCOREP_EXPERIMENT_DIRECTORY=scorep-b_eff_io-4-profile+posix
% export SCOREP_IO_POSIX=yes
# run with 4 tasks and 6 threads each ./run.sh b_eff_io
% cube scorep-b_eff_io-4-profile+posix/profile.cubex
```

# Mastering File I/O: Result visualization



# Mastering File I/O: Result visualization



# Mastering Python

- Install Score-P Python bindings via  
`pip install --user scorep`
- Execute Score-P as a module via  
`python -m scorep ...`
- Score-P environment variables are still valid and needed

```
#!/bin/env bash  
export SCOREP_EXPERIMENT_DIRECTORY=...  
export SCOREP_FILTERING_FILE=...  
python -m scorep <path/to/my_script.py>
```

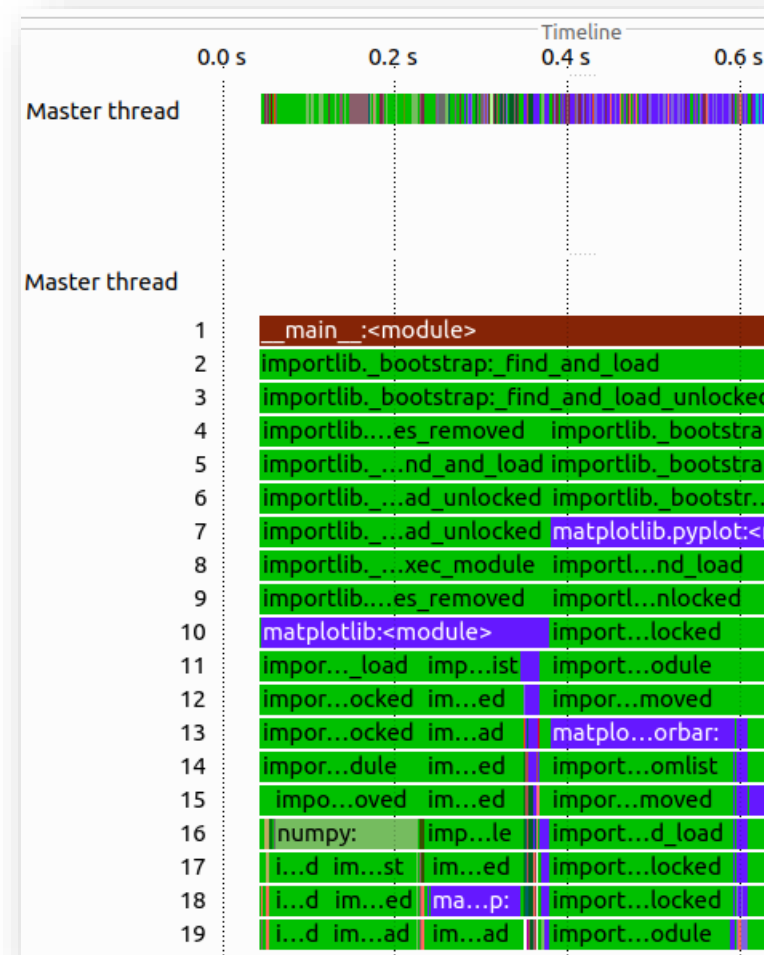


Fig.: Vampir – module import

# Mastering Python: Score-P options

- Accepts additional switches `-m scorep <options>`
- These include
  - `--thread=pthread`
  - `--mpp=mpi`
  - `--cuda / --hip`

```
#!/bin/env bash
```

```
python -m scorep <path/to/my_script.py>
```

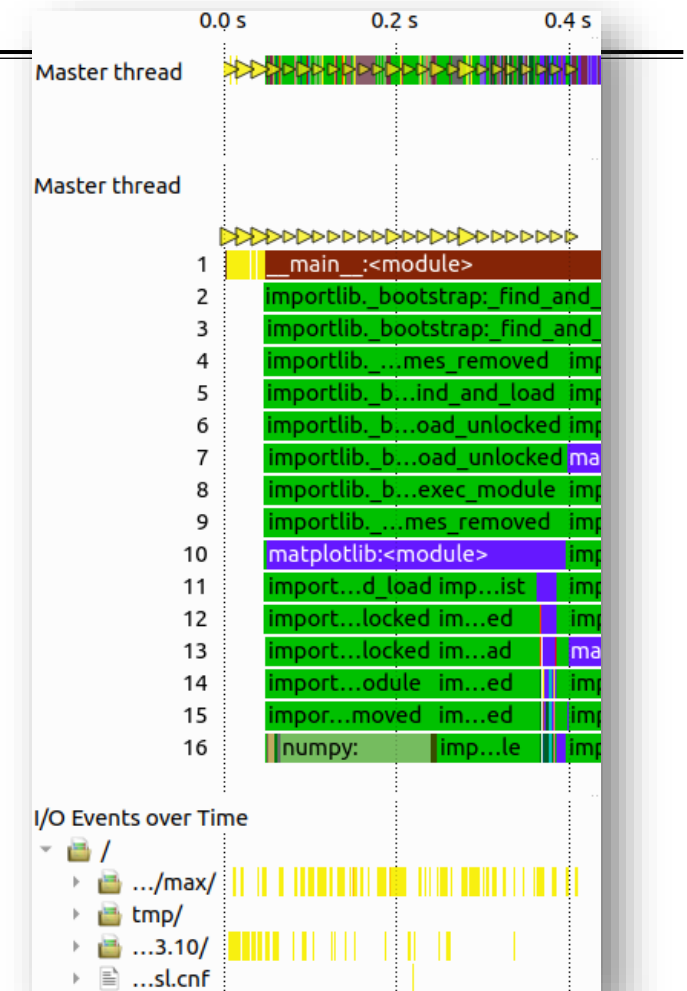


Fig.: Trace with POSIX-I/O

# Mastering Python: Selective instrumentation

- Enable/Disable instrumentation for certain functions
- Via decorator or context

```
import scorep

@scorep.instrumenter.enable()/disable()

with scorep.instrumenter.enable()/disable():
```
- [No] data is collected for these sections
- Disabling is useful for
  - Import of modules
  - Known uninteresting functions
  - Very 'hot' functions to reduce overhead
- Hint: name 'disabled' blocks for attribution in output

```
%%file 02_matmul.py
import scorep
with scorep.instrumenter.disable():
    import numpy as np
...

@scorep.instrumenter.disable()
def add(a,b)
    return a + b

def compute_diagonal_sum(A):
    sum_diag = 0
    for i in range(min(A.shape[0], A.shape[1])):
        sum_diag = add(sum_diag,A[i][i])
    return sum_diag

def main():
    A = generate_random_matrix(64, 64)
    diag_sum = compute_diagonal_sum(A)
...

```

# Mastering Python: Selective instrumentation

---

- Selective enabling of functions
- If only certain region is of interest
- Make sure to execute Score-P with

```
--noinstrumenter
```

```
#!/bin/env bash  
...  
python -m scorep --noinstrumenter \  
    02_matmul.py
```

```
%%file 02_matmul.py  
import scorep  
import numpy as np  
...  
def add(a,b)  
    return a + b  
def compute_diagonal_sum(A):  
    sum_diag = 0  
    for i in range(min(A.shape[0], A.shape[1])):  
        sum_diag = add(sum_diag,A[i][i])  
    return sum_diag  
  
def main():  
    A = generate_random_matrix(64, 64)  
    with scorep.instrumenter.enable():  
        diag_sum = compute_diagonal_sum(A)  
...  
...  
...
```

# Mastering Python: Selective instrumentation hands-on

---

- Run application with

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep-python-matmul
% srun ... python -m scorep ./02_matmul.py
% cube scorep-python-matmul/profile.cubex
```

- Use selective instrumentation

```
% <editor> 02_matmul.py
# comment-in "scorep" statements and "fix" indentation
% export SCOREP_EXPERIMENT_DIRECTORY=scorep-python-matmul+selective
% srun ... python -m scorep ./02_matmul.py
% cube scorep-python-matmul+selective/profile.cubex
```

# Mastering Python: MPI

---

- MPI must be enabled explicitly via `--mpp=mpi`

```
srun -np 4 python -m scorep --mpp=mpi <path/to/my_script.py>
```

- Caveats

- No support for MPI\_Mprobe/Mrecv, default for mpi4py -> disable usage
- No support for MPI\_THREAD\_MULTIPLE, default for mpi4py -> disable usage

```
#!/bin/env bash
```

```
# disables the use of MPI_Mprobe & MPI_Mrecv
```

```
export MPI4PY_RC_RECV_MPROBE=False
```

```
# Promise the MPI Library (and Score-P) that the funneled mode is
```

```
# used at most
```

```
export MPI4PY_RC_THREAD_LEVEL=funneled
```

```
srun -np 4 python -m scorep --mpp=mpi <path/to/my_script.py>
```

# Mastering Python: MPI hands-on

---

- Run application with

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep-python-mpi
% export MPI4PY_RC_RECV_MPROBE=False
% export MPI4PY_RC_THREAD_LEVEL=funneled
% srun ... python -m scorep --mpp=mpi ./03_mpi.py
% cube scorep-python-mpi/profile.cubex
```

# Mastering Python: Accelerators

---

- Score-P can be used to instrument accelerators (GPUs)
- `SCOREP_[CUDA|HIP]_ENABLE=yes` use default set of features

```
...  
export SCOREP_CUDA_ENABLE=yes  
export SCOREP_CUDA_BUFFER=1M # Increase if necessary  
  
python -m scorep --cuda <path/to/my_script.py>
```

```
...  
export SCOREP_HIP_ENABLE=yes  
export SCOREP_HIP_ACTIVITY_BUFFER_SIZE=1M # Increase if necessary  
  
python -m scorep --hip <path/to/my_script.py>
```

## Analysis report examination with Cube

---

Marc Schlütter  
Jülich Supercomputing Centre

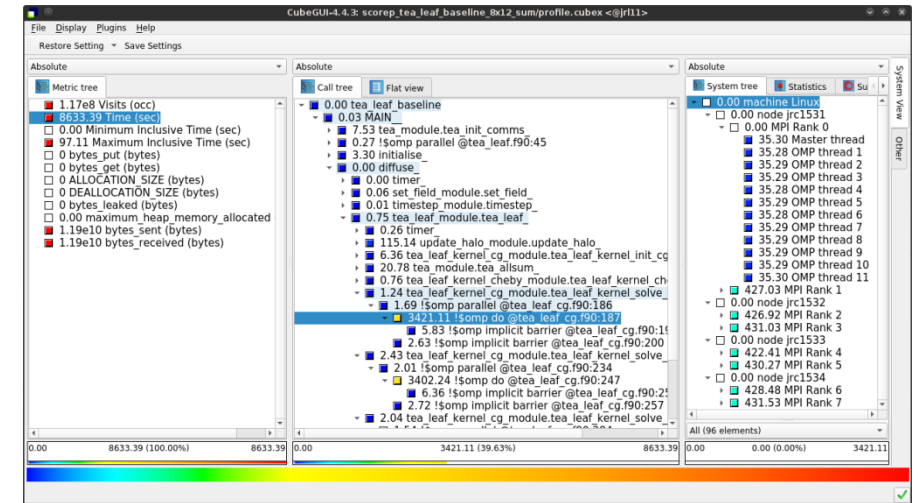


# Cube

CubeLib DOI 10.5281/zenodo.15051777

CubeGUI DOI 10.5281/zenodo.15051823

- 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  $\geq$  5
- Originally developed as part of the Scalasca toolset
- Now available as a separate components
  - Can be installed independently of Score-P, e.g., on laptop or desktop
  - Latest release: Cube v4.9.1 (December 2025)



**Note:** source distribution tarballs for Linux, as well as binary packages provided for Windows & MacOS, from [www.scalasca.org](http://www.scalasca.org) website in software/Cube-4x

# Cube GUI – Client Options & Usage

---

- Client Options:
  - User installation from tarball
  - **Running binary:**
    - win32 binary, Mac OS .dmg, Linux binary .AppImage
  - Available in Fedora distribution
  - POP3: Making Cube tools available on HPC Systems
  - JupyterLab/Webassembly integration
- Accessing Cube files
  - copy .cubex file (or entire scorep directory) to desktop from remote system via scp
  - **locally mount remote filesystem via sshfs**
  - Running a `cube_server` on the remote system and connect to it

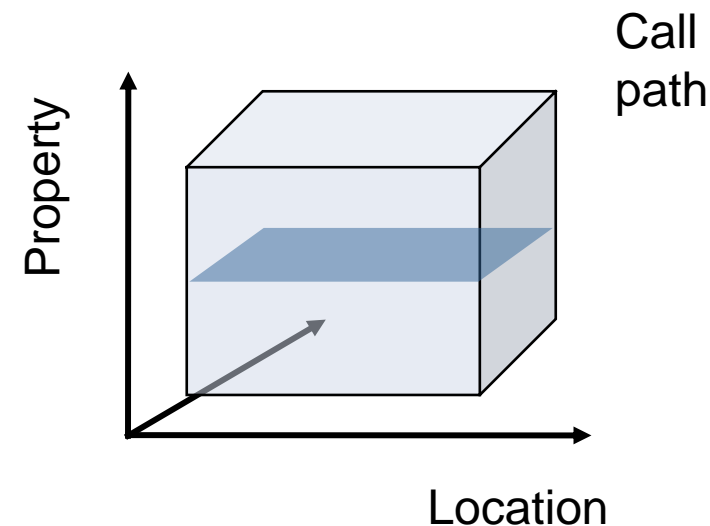
All download options:

<https://www.scalasca.org/scalasca/software/cube-4.x/download.html>

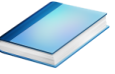
# Analysis presentation and exploration

---

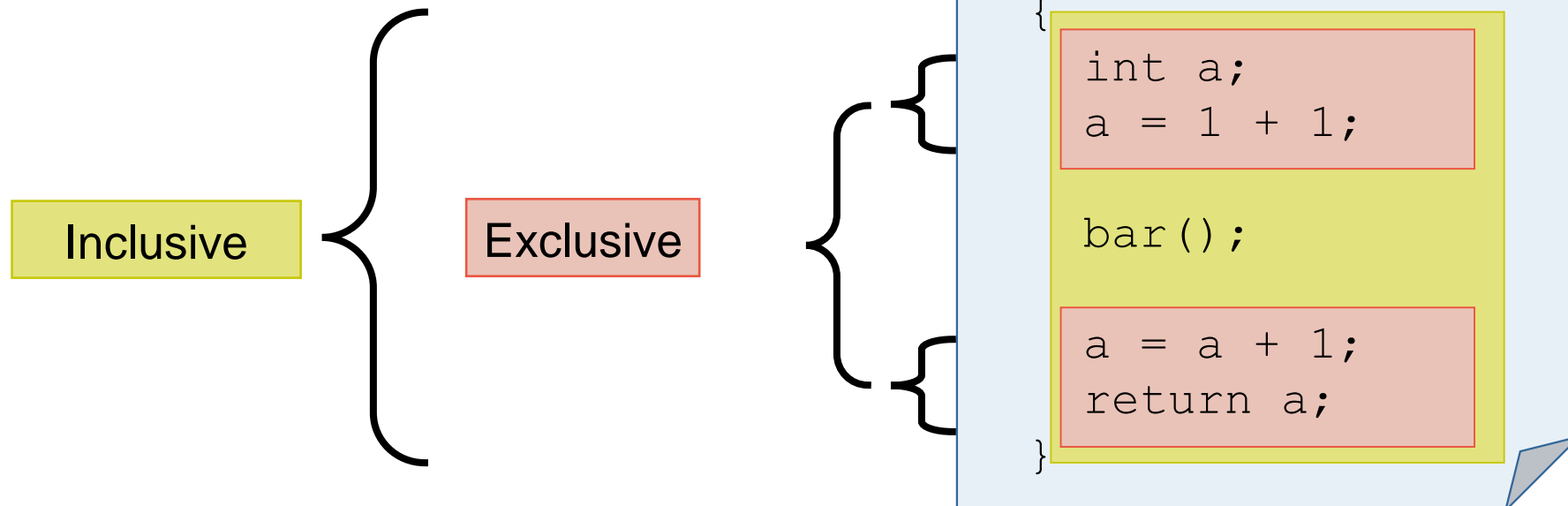
- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call path (program location)
  - System location (process/thread)
- Three coupled tree browsers
- Cube displays severities
  - *As value*: for precise comparison
  - *As colour*: for easy identification of hotspots
  - *Inclusive* value when closed & *exclusive* value when expanded
  - Customizable via display *modes*

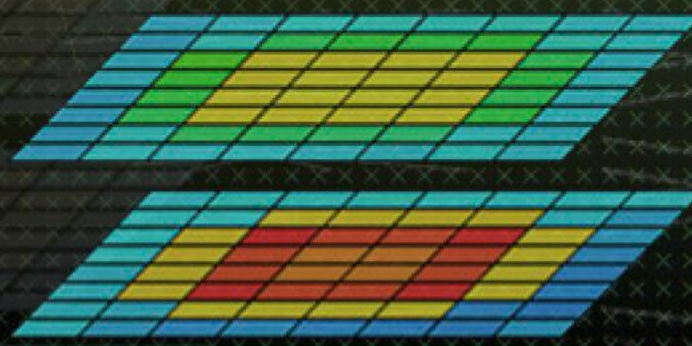


# Inclusive vs. exclusive values



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





## Demo: TeaLeaf case study

---



## Case study: TeaLeaf

---

- HPC mini-app developed by the UK Mini-App Consortium
  - Solves the linear 2D heat conduction equation on a spatially decomposed regular grid using a 5 point stencil with implicit solvers
  - Part of the Mantevo 3.0 suite
  - Available on GitHub: <http://uk-mac.github.io/TeaLeaf/>
  
- Measurements of TeaLeaf reference v1.0 taken on Jureca cluster @ JSC
  - Using Intel 19.0.3 compilers, Intel MPI 2019.3, and Score-P 5.0
  - Run configuration
    - 8 MPI ranks with 12 OpenMP threads each



```
% cd ~/workshop-vihps/Experiments
% cube scorep_tea_leaf_baseline_8x12_sum/profile.cubex
[GUI showing summary analysis report]
```

# Score-P analysis report exploration (opening view)

CubeGUI-4.4.3: scorep\_tea\_leaf\_baseline\_8x12\_sum/profile.cubex <@jrl11>

File Display Plugins Help

Restore Setting Save Settings

Absolute

Metric tree

- 1.17e8 Visits (occ)
- 8633.39 Time (sec)
- 0.00 Minimum Inclusive Time (sec)
- 97.11 Maximum Inclusive Time (sec)
- 0 bytes\_put (bytes)
- 0 bytes\_get (bytes)
- 0 ALLOCATION\_SIZE (bytes)
- 0 DEALLOCATION\_SIZE (bytes)
- 0 bytes\_leaked (bytes)
- 0.00 maximum\_heap\_memory\_allocated (bytes)
- 1.19e10 bytes\_sent (bytes)
- 1.19e10 bytes\_received (bytes)

Absolute

Call tree Flat view

- 1.17e8 tea leaf baseline

Absolute

System tree Statistics Sunburst Pr

- 1.17e8 machine Linux

System View Other

All (96 elements)

1.17e8 (100.00%) 1.17e8

What kind of performance metric?

Where is it in the source code?  
In what context?

How is it distributed across the processes/threads?

# Metric selection

CubeGUI-4.4.3: scorep\_tea\_leaf\_baseline\_8x12\_sum/profile.cubex <@jrl11>

File Display Plugins Help

Restore Setting Save Settings

Absolute

Metric tree

- 1.17e8 Visits (occ)
- 8633.39 Time (sec)
- 0.00 Minimum Inclusive Time (sec)
- 97.11 Maximum Inclusive Time (sec)
- 0 bytes\_put (bytes)
- 0 bytes\_get (bytes)
- 0 ALLOCATION\_SIZE (bytes)
- 0 DEALLOCATION\_SIZE (bytes)
- 0 bytes\_leaked (bytes)
- 0.00 maximum\_heap\_memory\_allocated (bytes)
- 1.19e10 bytes\_sent (bytes)
- 1.19e10 bytes\_received (bytes)

Absolute

Call tree Flat view

8633.39 tea leaf baseline

Absolute

System tree Statistics Sunburst Pr

8633.39 machine Linux

System View Other

All (96 elements)

0.00 8633.39 (100.00%) 8633.39

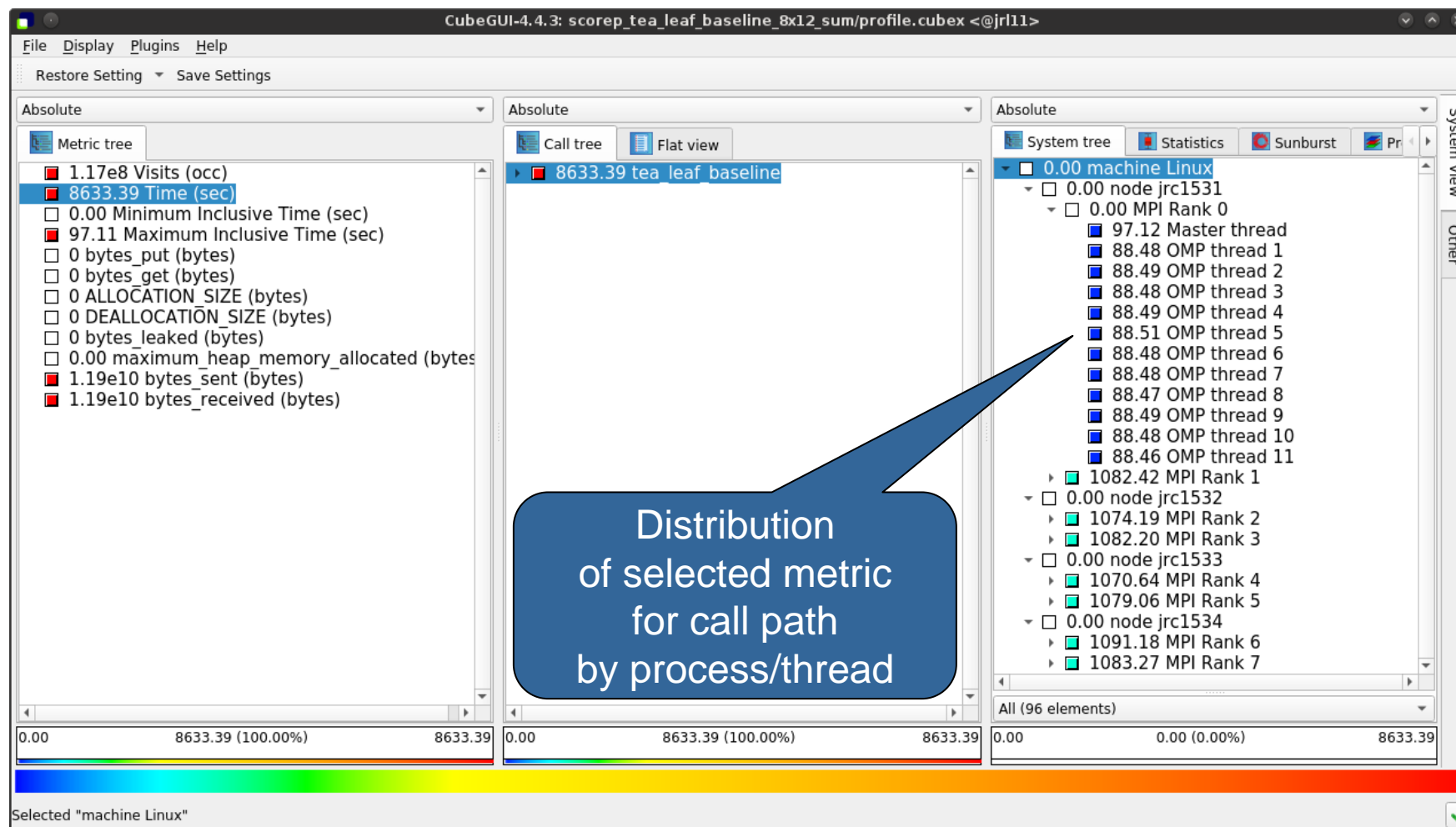
0.00 8633.39 (100.00%) 8633.39

0.00 8633.39 (100.00%) 8633.39

Selected "Time"

Selecting the "Time" metric shows total execution time

# Expanding the system tree



# Expanding the call tree

The screenshot displays the CubeGUI-4.4.3 interface for a performance profile. The main window is titled "scorep\_tea\_leaf\_baseline\_8x12\_sum/profile.cubex <@jrl11>". It features three main panels:

- Metric tree (left):** Shows a list of metrics. The "8633.39 Time (sec)" metric is selected and highlighted in blue. Other metrics include "1.17e8 Visits (occ)", "0.00 Minimum Inclusive Time (sec)", "97.11 Maximum Inclusive Time (sec)", and various memory-related metrics.
- Call tree (center):** Shows a hierarchical view of function calls. The root node is "0.03 MAIN\_". It is expanded to show several sub-nodes, including "7.53 tea\_module.tea\_init\_comms", "0.27 !\$omp parallel @tea\_leaf.f90:45", "3.30 initialise\_", "0.00 diffuse\_", "0.00 timer\_", "0.06 set\_field\_module.set\_field\_", "0.01 timestep\_module.timestep\_", "0.75 tea\_leaf\_module.tea\_leaf\_", "0.26 timer\_", "115.14 update\_halo\_module.update\_halo", "6.36 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_init\_cg", "20.78 tea\_module.tea\_allsum", "0.76 tea\_leaf\_kernel\_cheby\_module.tea\_leaf\_kernel\_ch", "1.24 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_solve\_", "1.69 !\$omp parallel @tea\_leaf\_cg.f90:186", "3421.11 !\$omp do @tea\_leaf\_cg.f90:187", "5.83 !\$omp implicit barrier @tea\_leaf\_cg.f90:191", "2.63 !\$omp implicit barrier @tea\_leaf\_cg.f90:200", "2.43 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_solve\_", "2.01 !\$omp parallel @tea\_leaf\_cg.f90:234", "3402.24 !\$omp do @tea\_leaf\_cg.f90:247", "6.36 !\$omp implicit barrier @tea\_leaf\_cg.f90:251", "2.72 !\$omp implicit barrier @tea\_leaf\_cg.f90:257", and "2.04 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_solve\_".
- System tree (right):** Shows the hardware configuration, including "0.00 machine Linux", "0.00 node jrc1531", "0.00 MPI Rank 0", and "0.00 MPI Rank 1" through "0.00 MPI Rank 7".

At the bottom of the call tree, a color bar indicates the time distribution. The total time is 8633.39 seconds. The call tree is expanded to show the inclusive value (8633.39) and the exclusive value (0.00). The callouts explain that the inclusive value is the sum of the inclusive values of all children, and the exclusive value is the sum of the exclusive values of all children.

**Distribution of selected metric across the call tree**

**Collapsed: inclusive value  
Expanded: exclusive value**

# Selecting a call path

The screenshot displays the CubeGUI-4.4.3 interface with three main panels:

- Metric tree (left):** Shows various performance metrics. The '8633.39 Time (sec)' metric is highlighted in blue.
- Call tree (middle):** Shows a hierarchical view of the program's execution. The path '0.75 tea\_leaf\_module.tea\_leaf' is expanded, and the sub-path '3421.11 !\$omp do @tea\_leaf\_cg.f90:187' is highlighted in blue.
- System tree (right):** Shows the system's hierarchical structure. The path '0.00 machine Linux' is expanded, and the sub-path '3421.11 MPI Rank 0' is highlighted in blue.

At the bottom, a color-coded bar indicates the relative contribution of each selected path to the total time. The bar is divided into three segments: a small blue segment (0.00), a large green segment (8633.39, 100.00%), and a small red segment (0.00). The total time is 8633.39 seconds.

A blue callout box with a white border and a pointer to the selected call path contains the text: "Selection updates metric values shown in columns to the right".

# Multiple selection

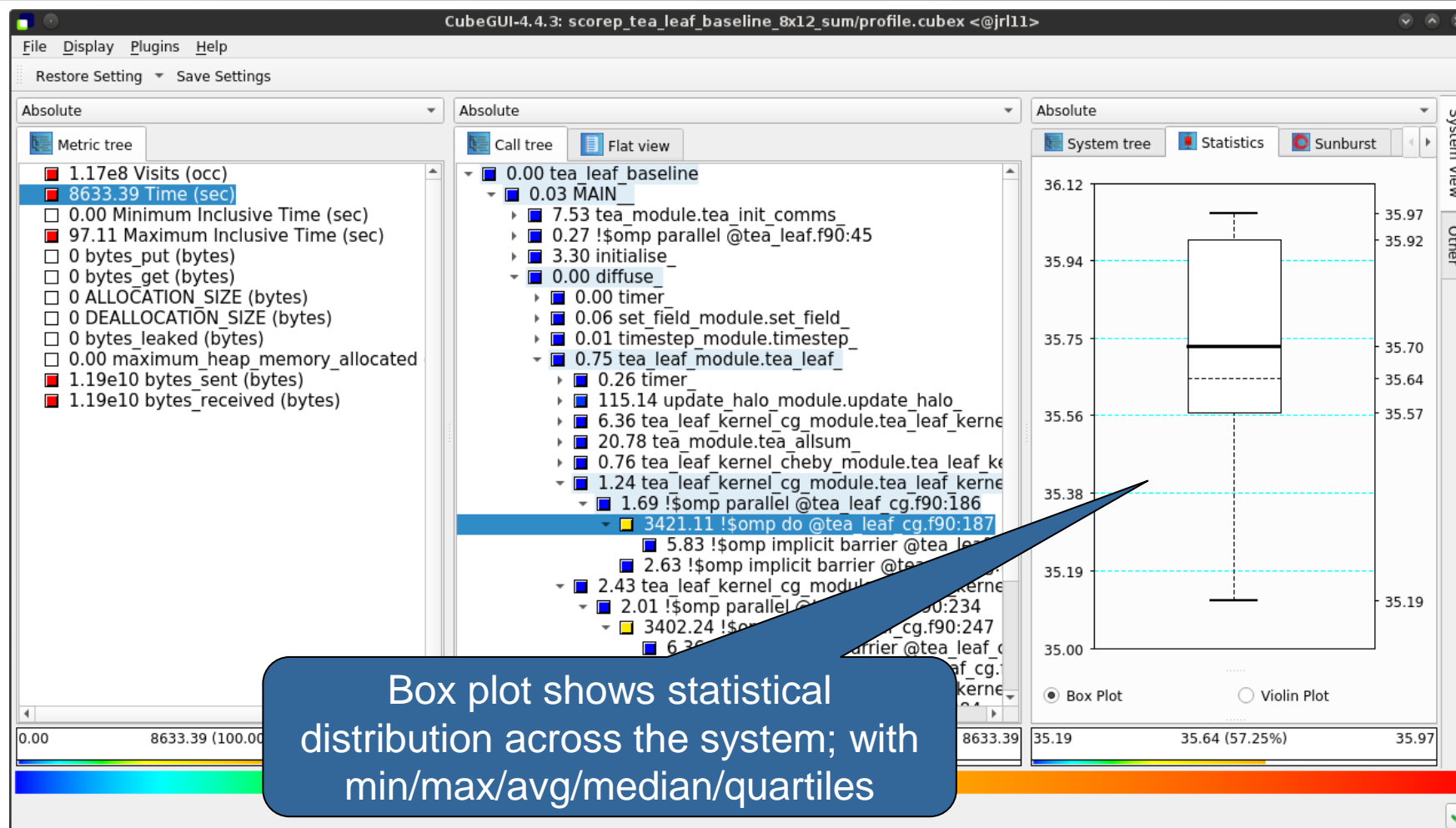
The screenshot displays the CubeGUI-4.4.3 interface for a performance profile. The window title is "CubeGUI-4.4.3: scorep\_tea\_leaf\_baseline\_8x12\_sum/profile.cubex <@jrl11>". The interface is divided into three main panels:

- Metric tree (left):** Shows various performance metrics. The "8633.39 Time (sec)" metric is highlighted in blue.
- Call tree (middle):** Shows a hierarchical view of the application's execution. Several nodes are selected with blue highlights, including:
  - 0.75 tea\_leaf\_module.tea\_leaf\_
  - 1.24 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_solve\_
  - 1.69 !\$omp parallel @tea leaf cg.f90:186
  - 3421.11 !\$omp do @tea leaf cg.f90:187
  - 5.83 !\$omp implicit barrier @tea leaf cg.f90:19
  - 2.63 !\$omp implicit barrier @tea leaf cg.f90:200
  - 2.43 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_solve\_
  - 2.01 !\$omp parallel @tea leaf cg.f90:234
  - 3402.24 !\$omp do @tea leaf cg.f90:247
  - 6.36 !\$omp implicit barrier @tea leaf cg.f90:25
  - 2.72 !\$omp implicit barrier @tea leaf cg.f90:257
  - 2.04 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_solve\_
  - 1.54 !\$omp parallel @tea leaf cg.f90:284
  - 1580.11 !\$omp do @tea leaf cg.f90:294
  - 40.82 !\$omp implicit barrier @tea leaf cg.f90:3
  - 3.24 !\$omp implicit barrier @tea leaf cg.f90:302
  - 1.37 tea\_leaf\_kernel\_module.tea\_leaf\_kernel\_finalise\_
  - 0.25 field\_summary\_
- System tree (right):** Shows the system hierarchy. The "0.00 machine Linux" node is expanded, showing multiple MPI ranks and threads. The "0.00 MPI Rank 0" node is expanded, showing 12 threads (Master thread, 11 OMP threads).

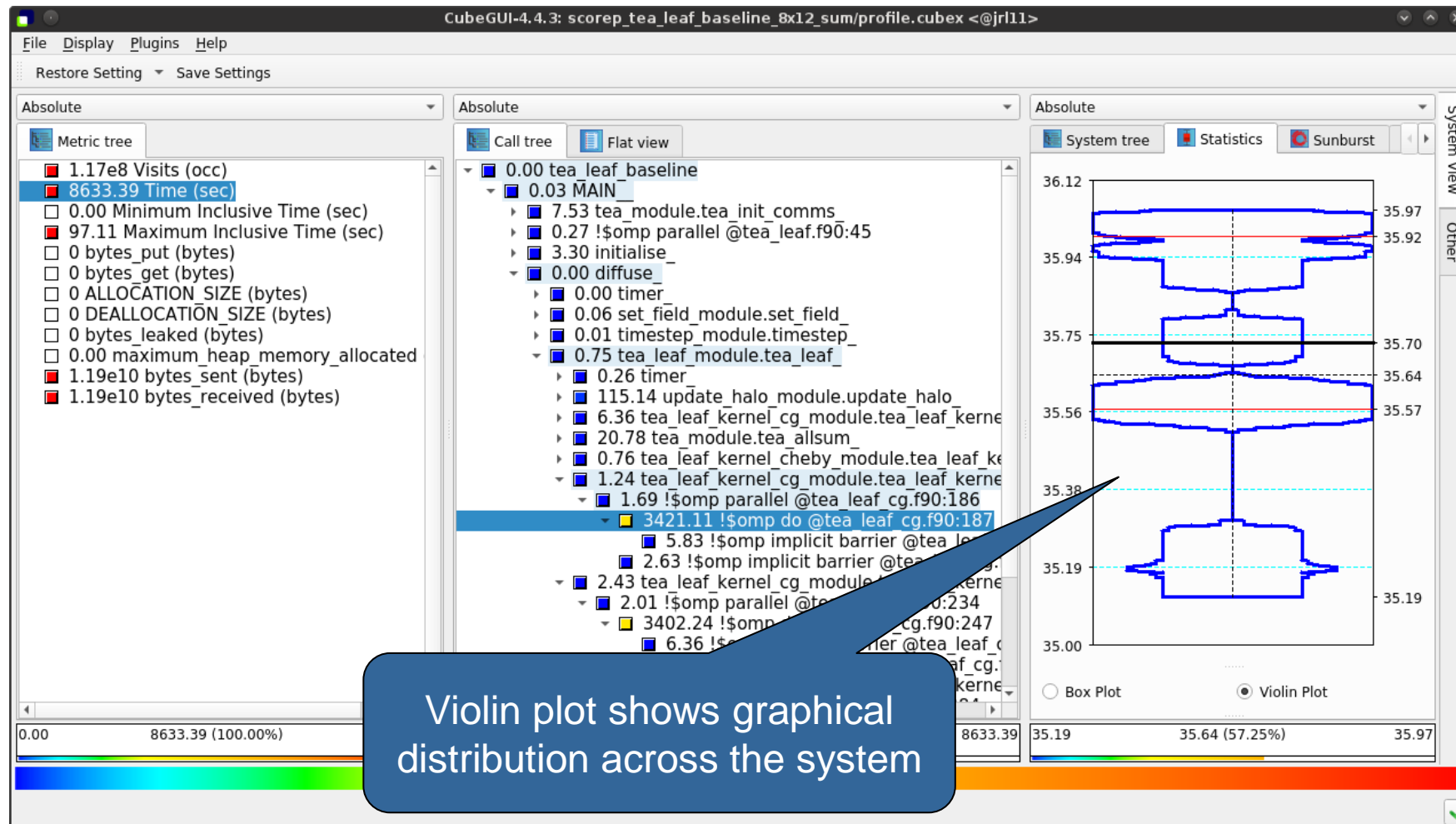
A blue callout box with a white border and a pointer to the selected nodes in the Call tree contains the text: "Select multiple nodes with Ctrl-click".

At the bottom of the interface, there are three progress bars showing the percentage of nodes selected in each view: Metric tree (100.00%), Call tree (97.34%), and System tree (0.00%).

# Box plot view



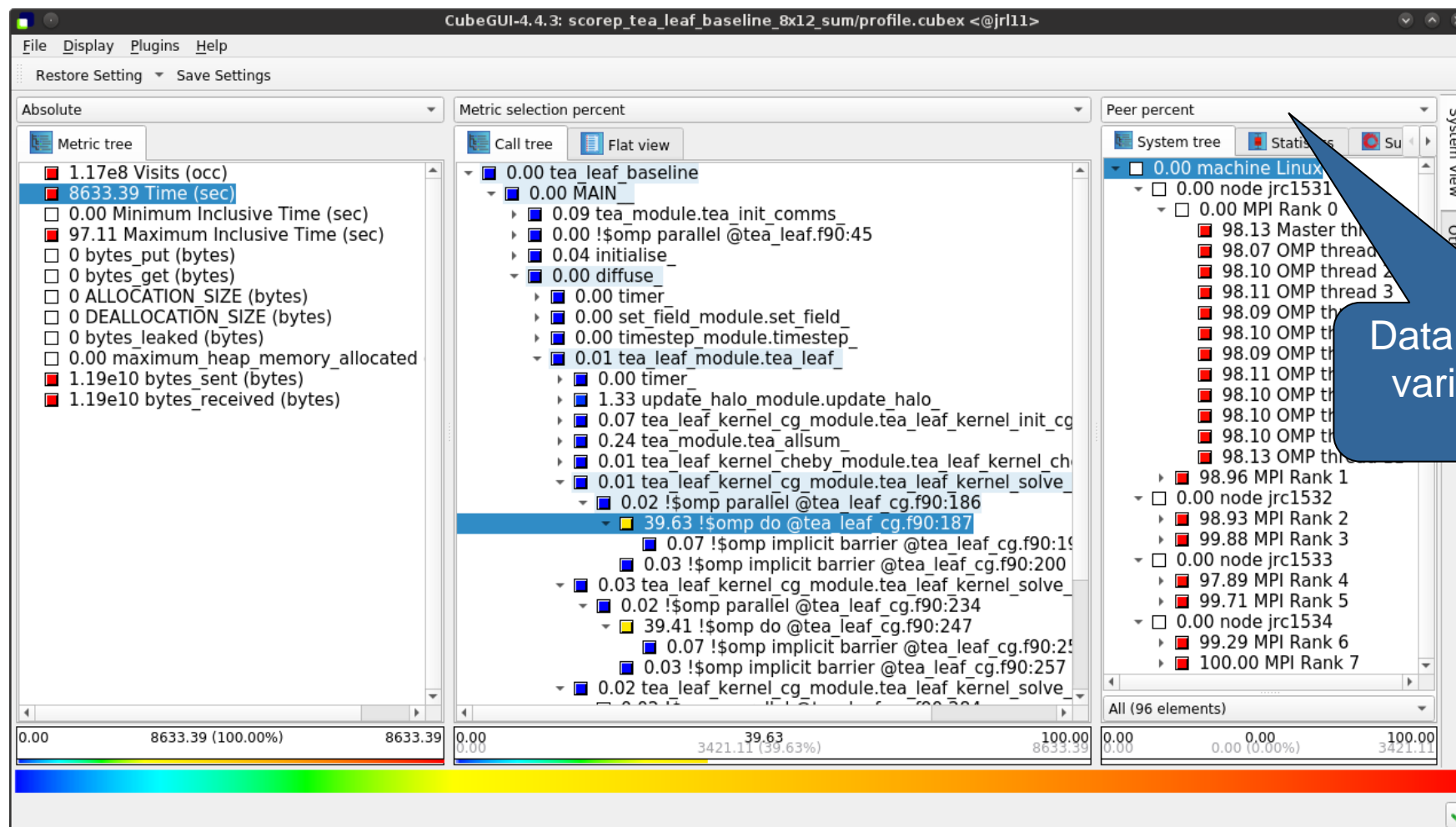
# Violin plot view







# Alternative display modes



Data can be shown in various percentage modes

# Important display modes

---

- Absolute
  - Absolute value shown in seconds/bytes/counts
  
- Selection percent
  - Value shown as percentage w.r.t. the selected node  
“on the left” (metric/call path)
  
- Peer percent (system tree only)
  - Value shown as percentage relative to the maximum peer value

# Source-code view via context menu

The screenshot displays the CubeGUI-4.4.3 interface with three main panels: Metric tree, Call tree, and System tree. The Call tree panel is active, showing a hierarchical view of execution metrics. A context menu is open over the item `3421.11 !$omp do @tea_leaf_cg.f90:18`. The menu options include: Info, Documentation, Set as loop, Expand/collapse, Hiding, Cut call tree, Find items, Clear found items, Sort tree items..., Min/max values, Copy to clipboard, Show max severity information, and Mark this item. A blue callout box with a white border points to the context menu with the text "Right-click opens context menu".

Right-click opens context menu

# Source-code view

The screenshot shows the CubeGUI-4.4.3 interface with the following components:

- Metric tree (Left):** A list of performance metrics including '8633.39 Time (sec)', '97.11 Maximum Inclusive Time (sec)', and '1.19e10 bytes\_sent (bytes)'.
- Call tree (Center):** A hierarchical view of the program's execution, showing '0.00 tea\_leaf\_baseline' and '0.03 MAIN\_'. A specific node is highlighted with a blue bar.
- Source view (Right):** A code editor showing Fortran source code. The 'Source' tab is selected, and the code is displayed with line numbers from 170 to 204. A blue callout box points to this tab with the text 'Select "Source" tab'.

**Note:** This feature depends on the availability of the source code, as well as file and line number information provided by the instrumentation, i.e., it may not always be available

# Context-sensitive help

The screenshot displays the CubeGUI-4.4.3 interface with the 'Help' menu open. The 'What's This?' option is selected, and a blue callout box points to it with the text: 'Context-sensitive help available for all GUI items'. The main window shows a hierarchical tree of metrics and system components. The selected metric is '39.63 !\$omp do @tea\_leaf\_cg.f90:187'. The interface includes a 'Metric tree' on the left, a central 'Flat view' of the selected metric, and a 'System tree' on the right. A color bar at the bottom indicates the range of values for the selected metric, from 0.00 to 8633.39.

File Display Plugins Help

Restore Setting Sa

Absolute

Metric tree

1.17e8 Visits (

8633.39 Time

0.00 Minimum

97.11 Maximum

0 bytes\_put (b

0 bytes\_get (b

0 ALLOCATE

0 DEALLOCATE

0 DEALLOCATE

0 heap\_memory\_allocated

0 bytes\_sent (bytes)

0 bytes\_received (bytes)

Getting started

User Guide

Mouse and keyboard control

What's This? Shift+F1

About

Plugin Info

Plugin Documentation

Selected metrics description

Selected regions description

tea\_leaf\_baseline

0 MAIN\_

0.09 tea\_module.tea\_init\_comms

0.00 !\$omp parallel @tea\_leaf.f90:45

0.04 initialise\_

0.00 diffuse\_

0.00 timer\_

0.00 set\_field\_module.set\_field\_

0.00 timestep\_module.timestep\_

0.01 tea\_leaf\_module.tea\_leaf\_

0.00 timer\_

1.33 update\_halo\_module.update\_halo\_

0.07 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_init\_cg\_

0.24 tea\_module.tea\_allsum\_

0.01 tea\_leaf\_kernel\_cheby\_module.tea\_leaf\_kernel\_ch\_

0.01 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_solve\_

0.02 !\$omp parallel @tea\_leaf\_cg.f90:186

39.63 !\$omp do @tea\_leaf\_cg.f90:187

0.07 !\$omp implicit barrier @tea\_leaf\_cg.f90:19

0.03 !\$omp implicit barrier @tea\_leaf\_cg.f90:200

0.03 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_solve\_

0.02 !\$omp parallel @tea\_leaf\_cg.f90:234

39.41 !\$omp do @tea\_leaf\_cg.f90:247

0.07 !\$omp implicit barrier @tea\_leaf\_cg.f90:25

0.03 !\$omp implicit barrier @tea\_leaf\_cg.f90:257

0.02 tea\_leaf\_kernel\_cg\_module.tea\_leaf\_kernel\_solve\_

0.00

8633.39 (100.00%)

8633.39

0.00

0.00

39.63

100.00

8633.39

0.00

0.00

0.00 (0.00%)

100.00

3421.11

System tree

Statistics

System View

Other

0.00 machine Linux

0.00 node jrc1531

0.00 MPI Rank 0

98.13 Master thread

98.07 OMP thread 1

98.10 OMP thread 2

98.11 OMP thread 3

98.09 OMP thread 4

98.10 OMP thread 5

98.09 OMP thread 6

98.11 OMP thread 7

98.10 OMP thread 8

98.10 OMP thread 9

98.10 OMP thread 10

98.13 OMP thread 11

98.96 MPI Rank 1

0.00 node jrc1532

98.93 MPI Rank 2

99.88 MPI Rank 3

0.00 node jrc1533

97.89 MPI Rank 4

99.71 MPI Rank 5

0.00 node jrc1534

99.29 MPI Rank 6

100.00 MPI Rank 7

All (96 elements)

Change into help mode for display components

## Scalasca report post-processing

---

- Scalasca's report post-processing derives additional metrics and generates a structured metric hierarchy
- Automatically run (if needed) when using the **square** convenience command:

```
% square scorep_tea_leaf_baseline_8x12_sum  
INFO: Post-processing runtime summarization report (profile.cubex)...  
INFO: Displaying ./scorep_tea_leaf_baseline_8x12_sum/summary.cubex...
```

```
[GUI showing post-processed summary analysis report]
```

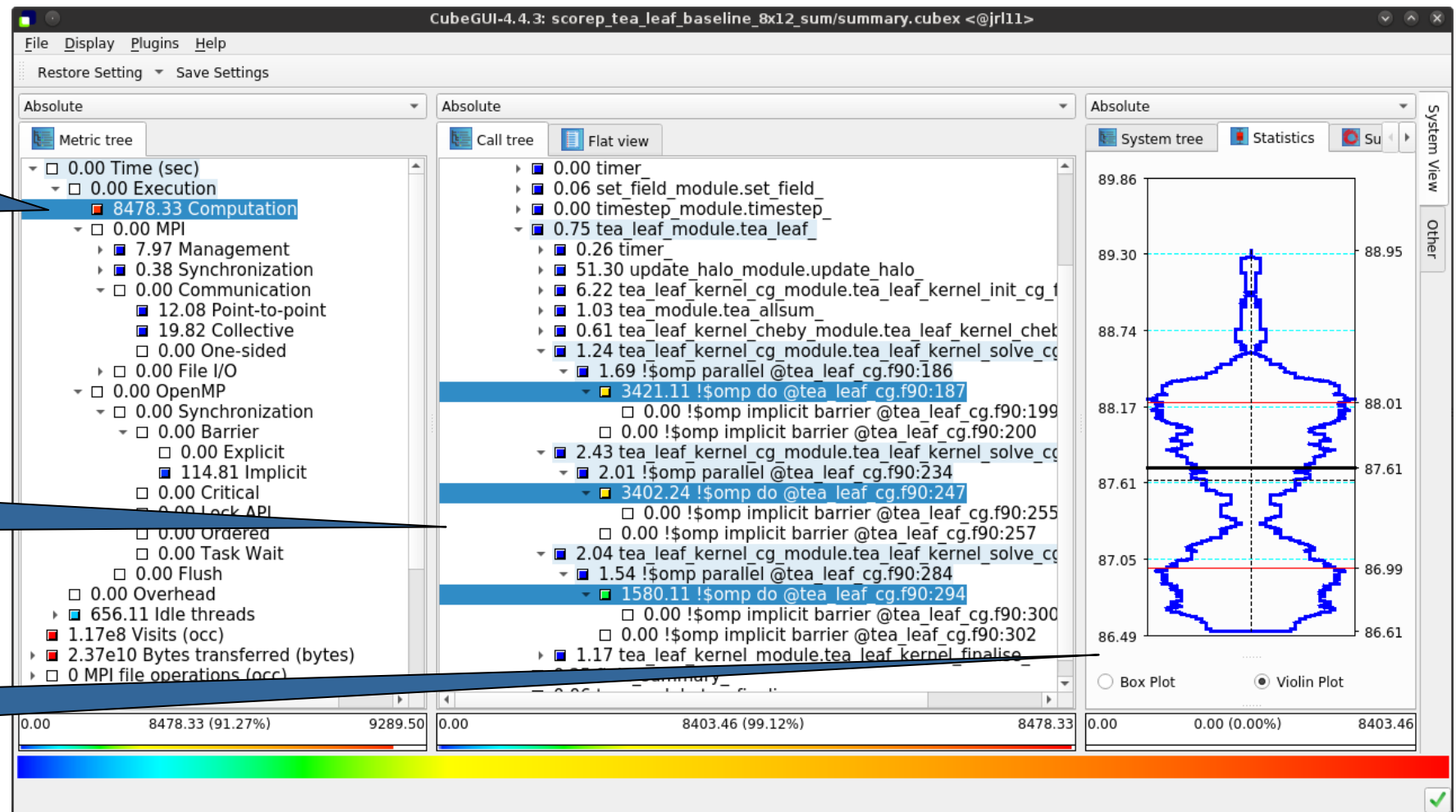


# TeaLeaf summary report analysis (I)

91% of the execution time is computation...

...almost entirely spent in 3 OpenMP do loops...

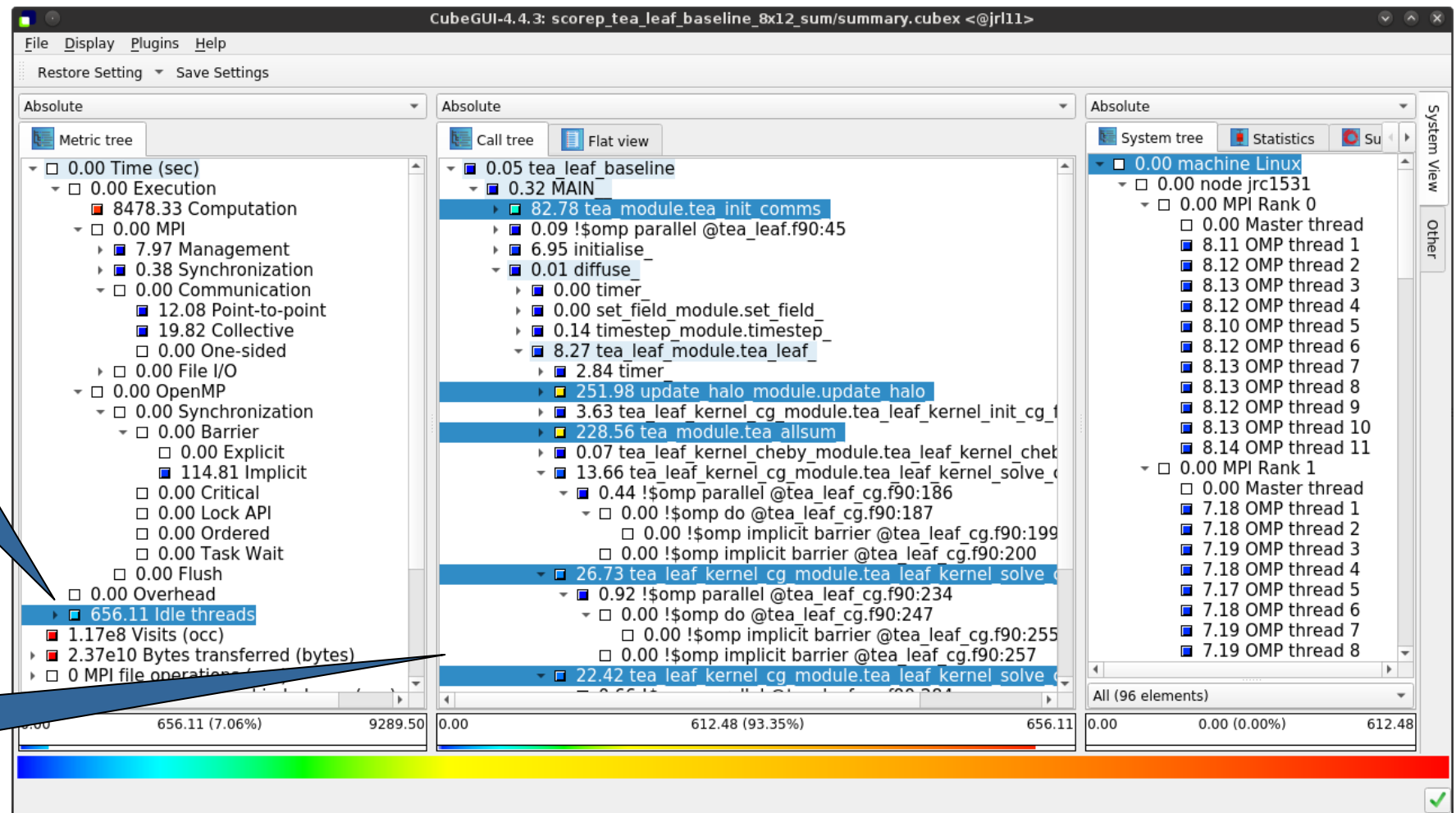
...with a slight imbalance across ranks & threads



# TeaLeaf summary report analysis (II)

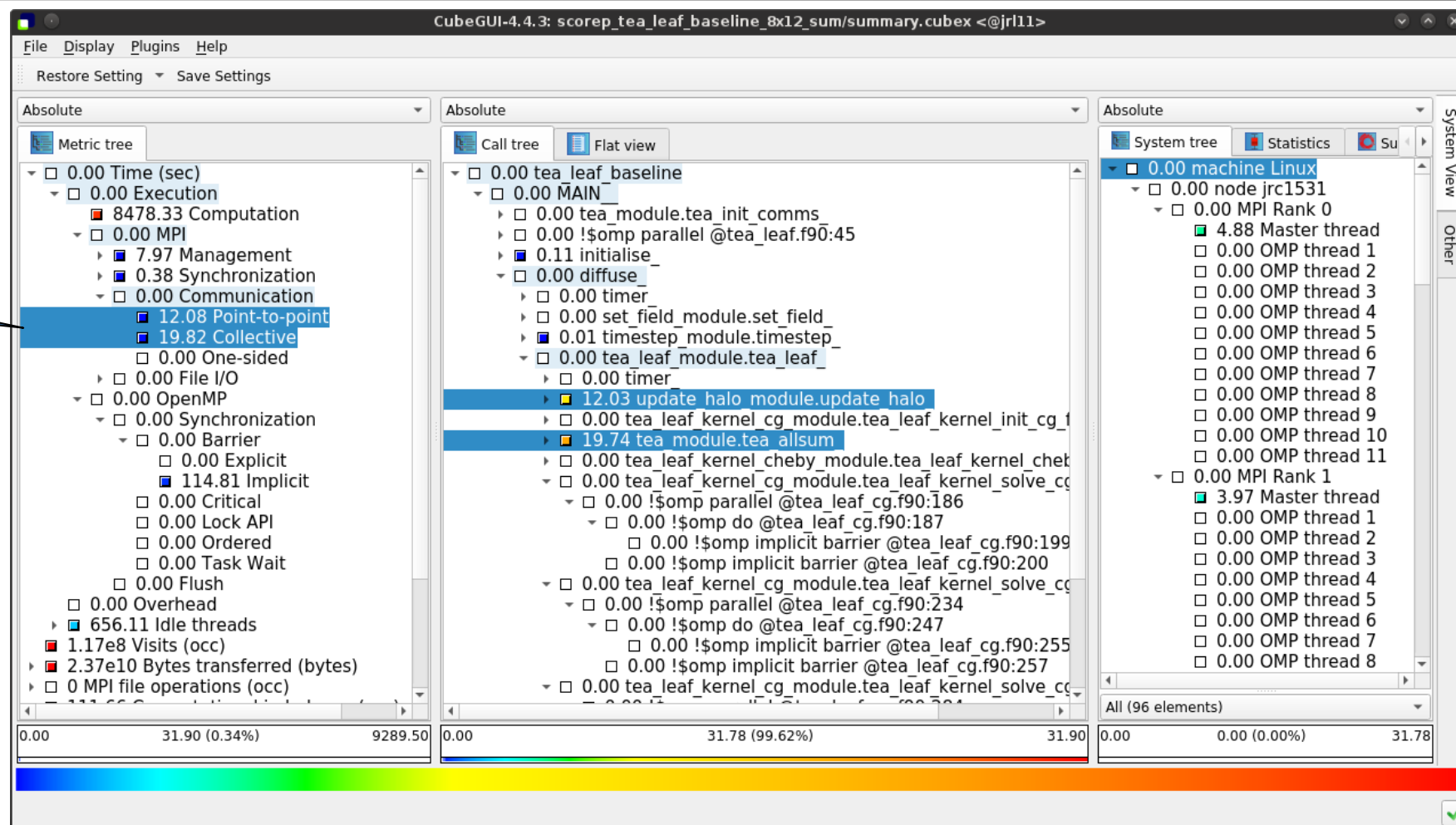
7% of the total CPU execution time is due due to "idle threads" ...

... when not within OpenMP-parallelized code regions



# TeaLeaf summary report analysis (III)

MPI communication time is negligible (0.34%); but communication is only on the master threads (MPI\_THREAD\_FUNNELED)

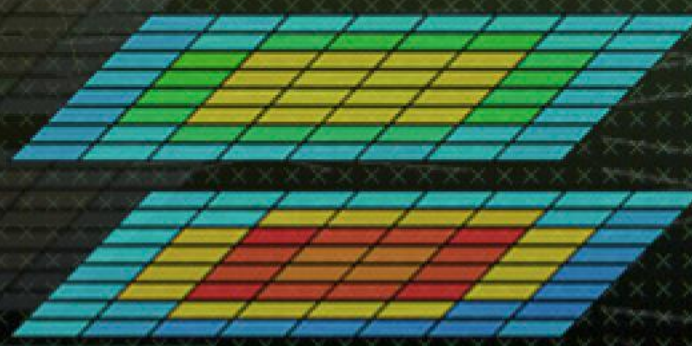


## Cube: Further information

---

- Parallel program analysis report exploration tools
  - Libraries for Cube report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
- Available under 3-clause BSD open-source license
- Documentation & sources:
  - <https://www.scalasca.org>
- User guide also part of installation:
  - `<prefix>/share/doc/cubegui/CubeUserGuide.pdf`
- Contact:
  - mailto: [scalasca@fz-juelich.de](mailto:scalasca@fz-juelich.de)

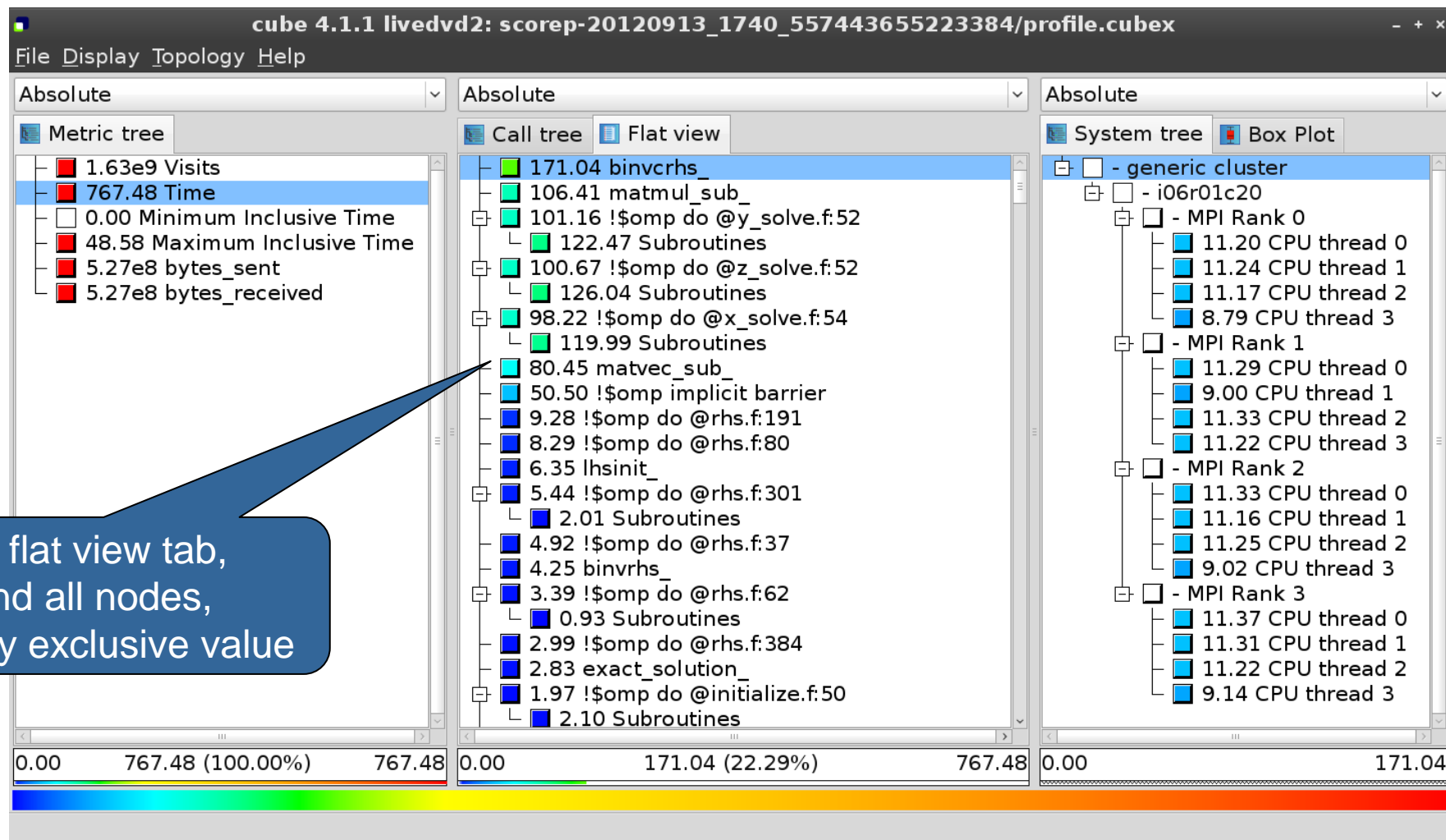




## Reference material



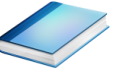
# Flat profile view



Select flat view tab,  
expand all nodes,  
and sort by exclusive value

## Derived metrics

---



- Derived metrics are defined using CubePL expressions, e.g.:

**metric::time(i)/metric::visits(e)**

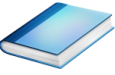
- Values of derived metrics are not stored, but calculated on-the-fly
- Types of derived metrics:
  - Prederived: evaluation of the CubePL expression is performed before aggregation
  - Postderived: evaluation of the CubePL expression is performed after aggregation

- Examples:

- “Average execution time”: Postderived metric with expression

**metric::time(i)/metric::visits(e)**

# Derived metrics in Cube GUI



Collection of derived metrics

Parameters of the derived metric

CubePL expression

Cube-4.3.1: scorep-20140130\_2053\_439988765917/profile.cubex

File Display Plugins Help

Metric tree

- 1.09e8 Visits (occ)
- 1.01e6 Time (sec)
- 0.00 Minimum Inclusive Time (sec)
- 246.14 Maximum Inclusive Time (sec)
- 7.18e12 bytes\_sent
- 7.18e12 bytes\_received

Call tree

- 0.35 main(int, char \*)
- 2512.10 ugshellInit
- 1.01e6 ug::script::LoadUGScript(const char \*, bool)
- 2.11 ug::script::ParseBuffer(const char \*, const char \*)
- 0.04 ugshellFinalize
- 94.31 MPI\_Finalize

System tree

- machine Blue Gene/Q
- rack 11
- midplane 1
- nodeboard 8
- nodecard 4
- 0.65 MPI Rank 0
- 0.64 MPI Rank 1

Create new metric as a child of metric

Select metric from collection: Average execution time (kenobi)

Derived metric type: Postderived metric

Display name: Average visit time

Unique name: avg\_visit\_time

Data type: DOUBLE

Unit of measurement: sec

URL:

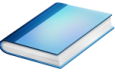
Description:  
Calculates average time of region execution per visit.  
Autor is Michael Knobloch.

Calculation  
metric::time()/metric::visits(e)

Create metric Cancel

Share this metric with SCALASCA group

# Example: FLOPS based on PAPI\_FP\_OPS and time



Cube-4.3.1: scorep\_8x4\_sum/profile.cubex (on froggy1)

File Display Plugins Help  
Restore Setting Save Settings

**Edit metric FLOPS (on froggy1)**

Select metric from collection: --- please select ---

Derived metric type: Postderived metric

Display name: FLOPS

Unique name: flops

Data type: DOUBLE

Unit of measurement:

URL:

Description:

Calculation Calculation Init Aggregation "+" Aggregation "-"

`metric::PAPI_FP_OPS()/metric::time()`

Edit metric Cancel

Share this metric with SCALASCA group

**Absolute** Metric tree

- 1.17e7 Visits (occ)
- 1148.49 Time (sec)
- 0.00 Minimum Inclusive Time (sec)
- 41.57 Maximum Inclusive Time (...)
- 0 bytes\_put (bytes)
- 0 bytes\_get (bytes)
- 5.75e12 PAPI\_TOT\_INS (#)
- 2.69e12 PAPI\_TOT\_CYC (#)
- 2.12e12 PAPI\_FP\_OPS (#)
- 3.12e9 bytes\_sent (bytes)
- 3.12e9 bytes\_received (bytes)
- 1.84e9 FLOPS**

**Absolute** Call tree Flat view

- 3.17e5 MAIN\_
  - 7.04e5 mpi\_setup\_
    - 6.34e4 MPI\_Bcast
    - 2.05e5 env\_setup\_
      - 7.39e5 zone\_setup\_
        - 9.31e5 map\_zones\_
          - 9.39e4 zone\_starts\_
            - 6.16e5 set\_constants\_
              - 5.91e8 initialize\_
                - 0.00 exact\_rhs\_
                  - 145.62 !\$omp parallel @exac...
                    - 2.54e4 !\$omp do @exact\_r...
                      - 9.65e8 !\$omp do @exact\_r...**
                      - 9.62e8 !\$omp do @exact\_r...
                      - 8.14e8 !\$omp do @exact\_r...
                      - 1.21e5 !\$omp do @exact\_r...
                      - 0.00 !\$omp implicit barrier...
                    - 6.23e4 exch\_qbc\_
                      - 1.94e9 adi\_
                        - 2.19e5 MPI\_Barrier
                        - 1.92e9 <<bt\_iter>> (200 itera...
                        - 1.98e8 verify\_
                          - 1.05e5 MPI\_Reduce

**Absolute** System tree Barplot Heatmap

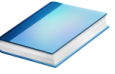
    - machine Linux
      - node frog6
        - MPI Rank 0
          - 1.17e9 Master thread
          - 9.43e8 OMP thread 1
          - 9.47e8 OMP thread 2
          - 9.47e8 OMP thread 3
        - MPI Rank 1
          - 1.17e9 Master thread
          - 9.87e8 OMP thread 1
          - 9.68e8 OMP thread 2
          - 9.72e8 OMP thread 3
        - MPI Rank 2
          - 1.10e9 Master thread
          - 8.97e8 OMP thread 1
          - 8.77e8 OMP thread 2
          - 8.76e8 OMP thread 3
        - MPI Rank 3
          - 1.09e9 Master thread
          - 9.06e8 OMP thread 1
          - 9.04e8 OMP thread 2
          - 9.02e8 OMP thread 3

All (32 elements)

0.00 1.84e9 (100.00%) 1.84e9 0.00 9.65e8 (-0.00%) -12858016489314434.00 0.00... -179769313486231570814527423731704356798070...

Selected "\$!omp do @exact\_rhs.f:46"

# CUBE algebra utilities



- Extracting solver sub-tree from analysis report

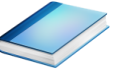
```
% cube_cut -r '<<ITERATION>>' scorep_bt-mz_C_32x4_sum/profile.cubex  
Writing cut.cubex... done.
```

- Calculating difference of two reports

```
% cube_diff scorep_bt-mz_C_32x4_sum/profile.cubex cut.cubex  
Writing diff.cubex... done.
```

- Additional utilities for merging, calculating mean, etc.
- Default output of `cube_utility` is a new report `utility.cubex`
- Further utilities for report scoring & statistics
- Run utility with ``-h`` (or no arguments) for brief usage info

# Iteration profiling



- Show time dependent behavior by “unrolling” iterations
- Preparations:
  - Mark loop body by using Score-P instrumentation API in your source code

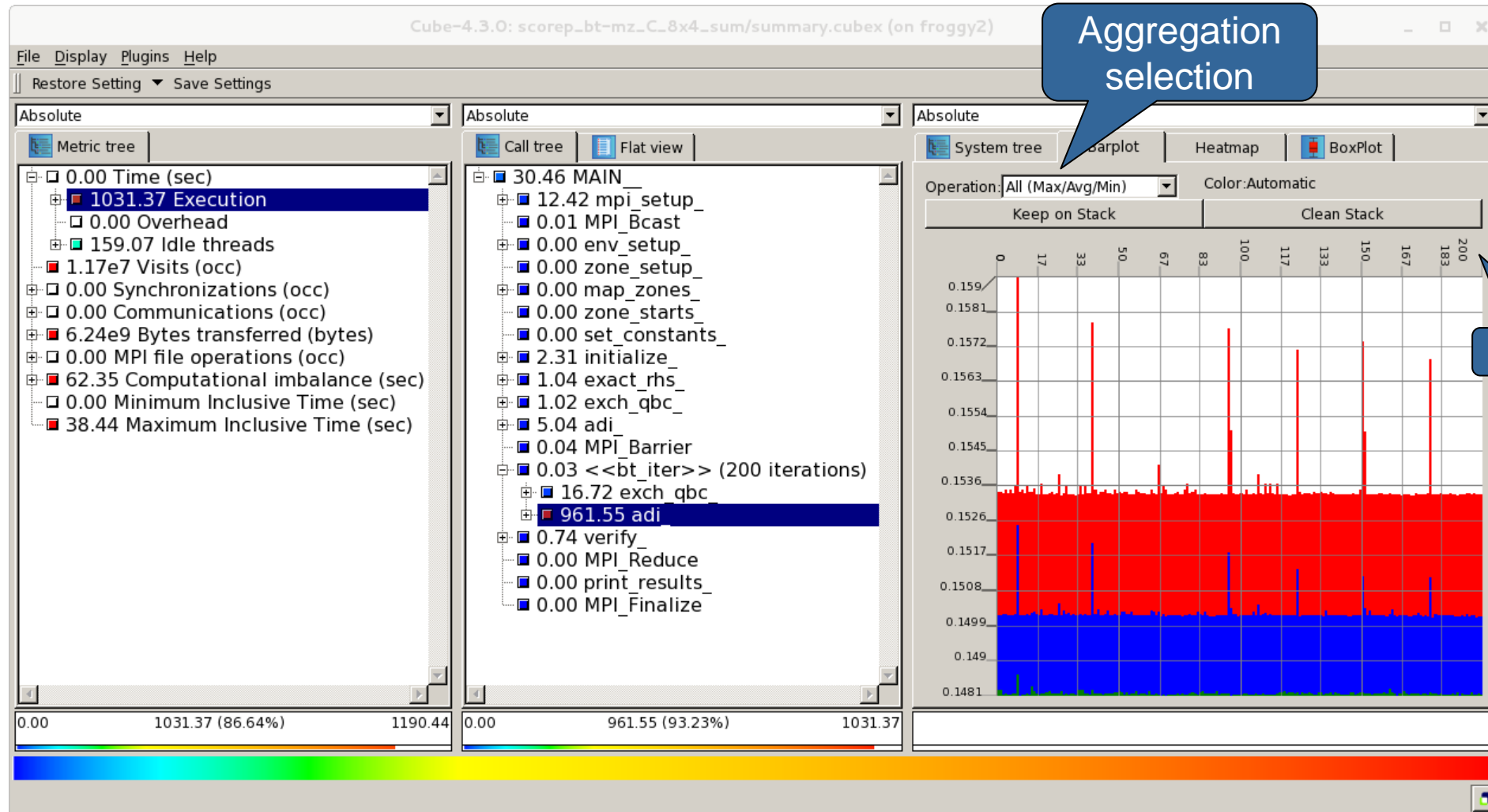
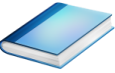
```
SCOREP_USER_REGION_DEFINE( scorep_bt_loop )  
SCOREP_USER_REGION_BEGIN( scorep_bt_loop, "<<bt_iter>>", SCOREP_USER_REGION_TYPE_DYNAMIC )  
SCOREP_USER_REGION_END( scorep_bt_loop )
```

- Result in the Cube profile:
  - Iterations shown as separate call trees
  - Useful for checking results for specific iterations

or

- Select your user-instrumented region and mark it as loop
- Choose “Hide iterations”
- View the Barplot statistics or the (thread x iterations) Heatmap

# Iteration profiling: Barplot



# Iteration profiling: Heatmap

