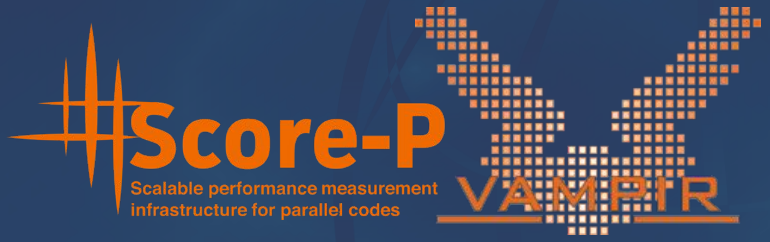
TECHNISCHE UNIVERSITÄT DRESDEN

Performance Analysis at Scale: The Score-P Tools Infrastructure



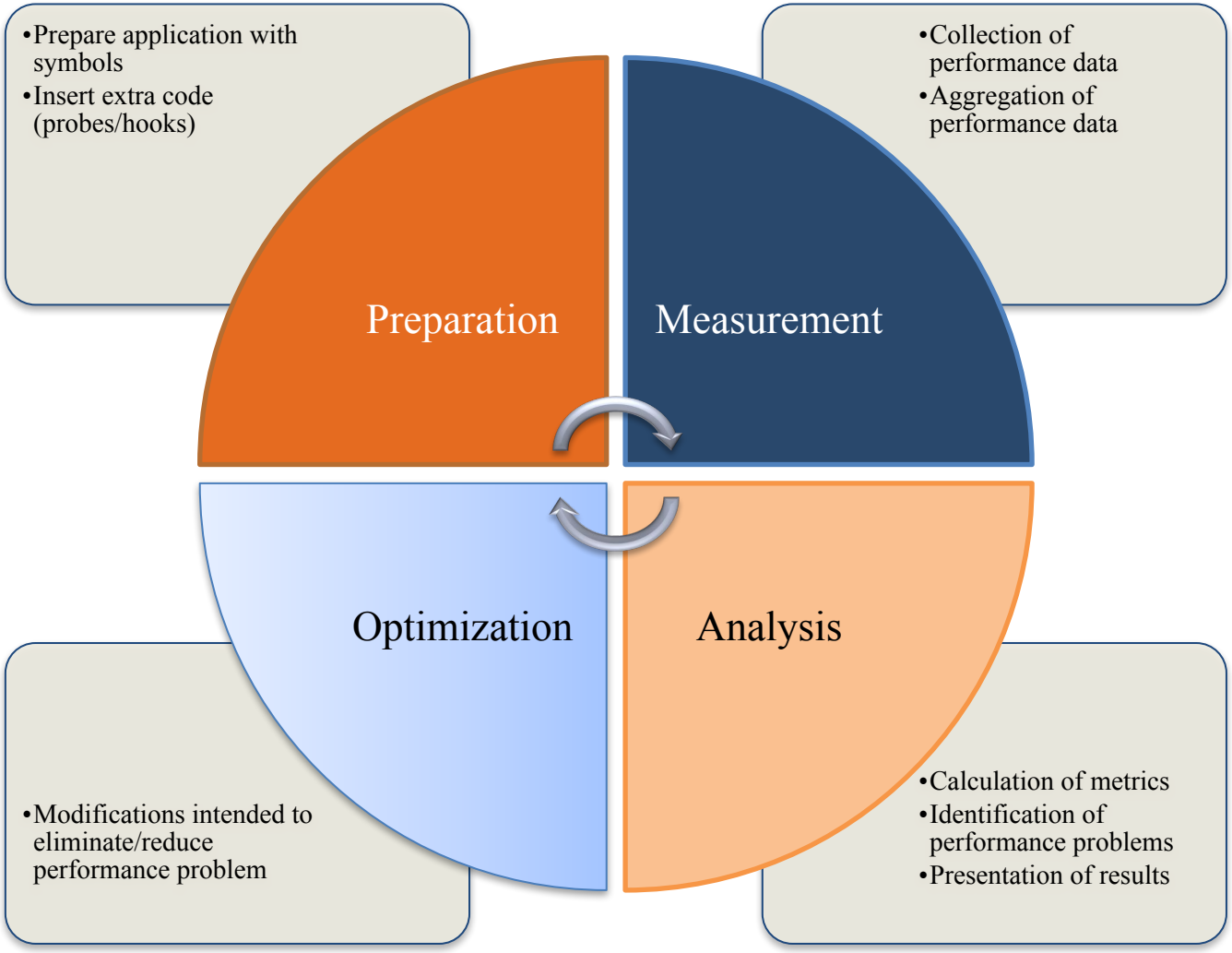
Frank Winkler (frank.winkler@tu-dresden.de)

ZIH
Center for Information Services & High Performance Computing

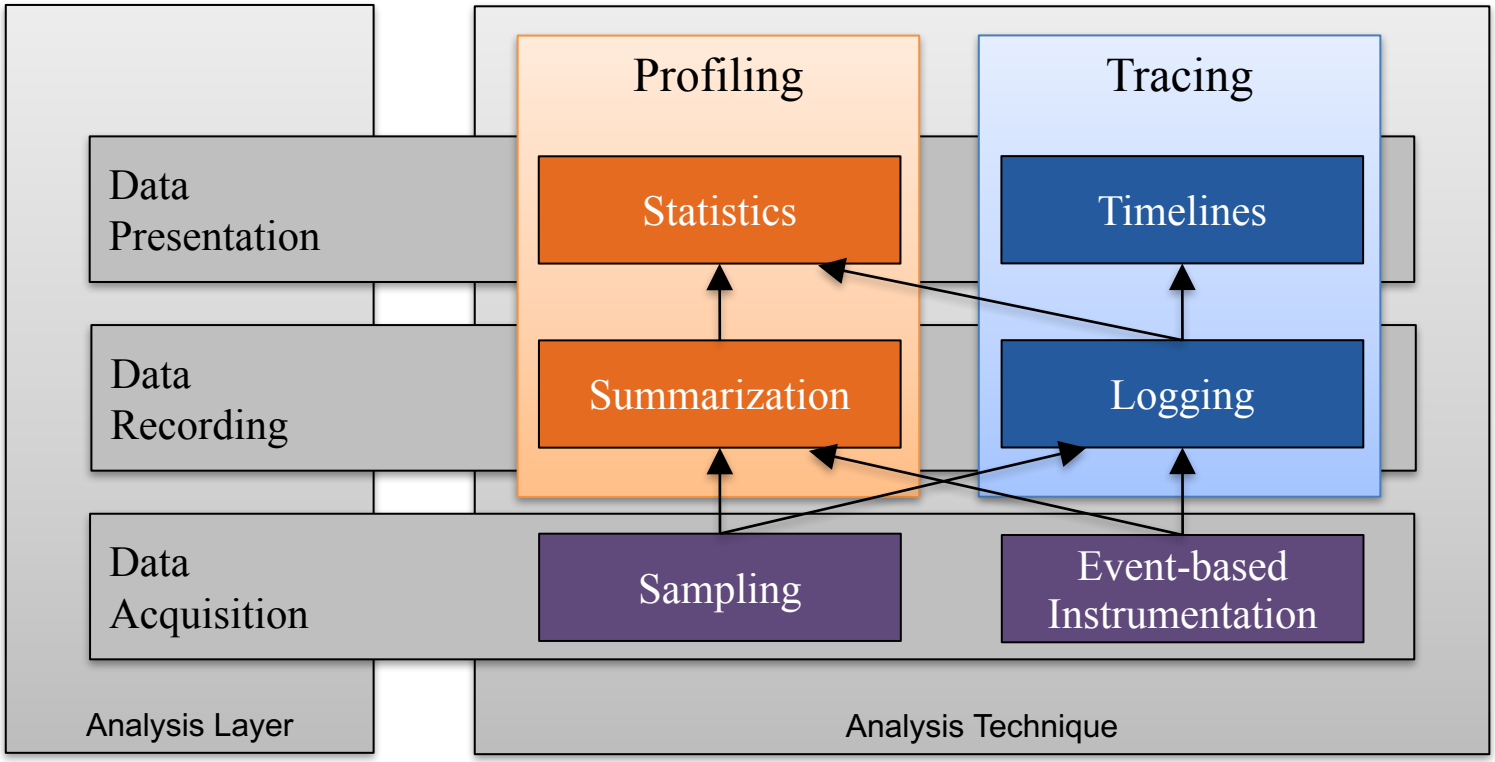
Disclaimer

Performance tools will not automatically make your code run faster. They help you understand, what your code does and where to put in work.

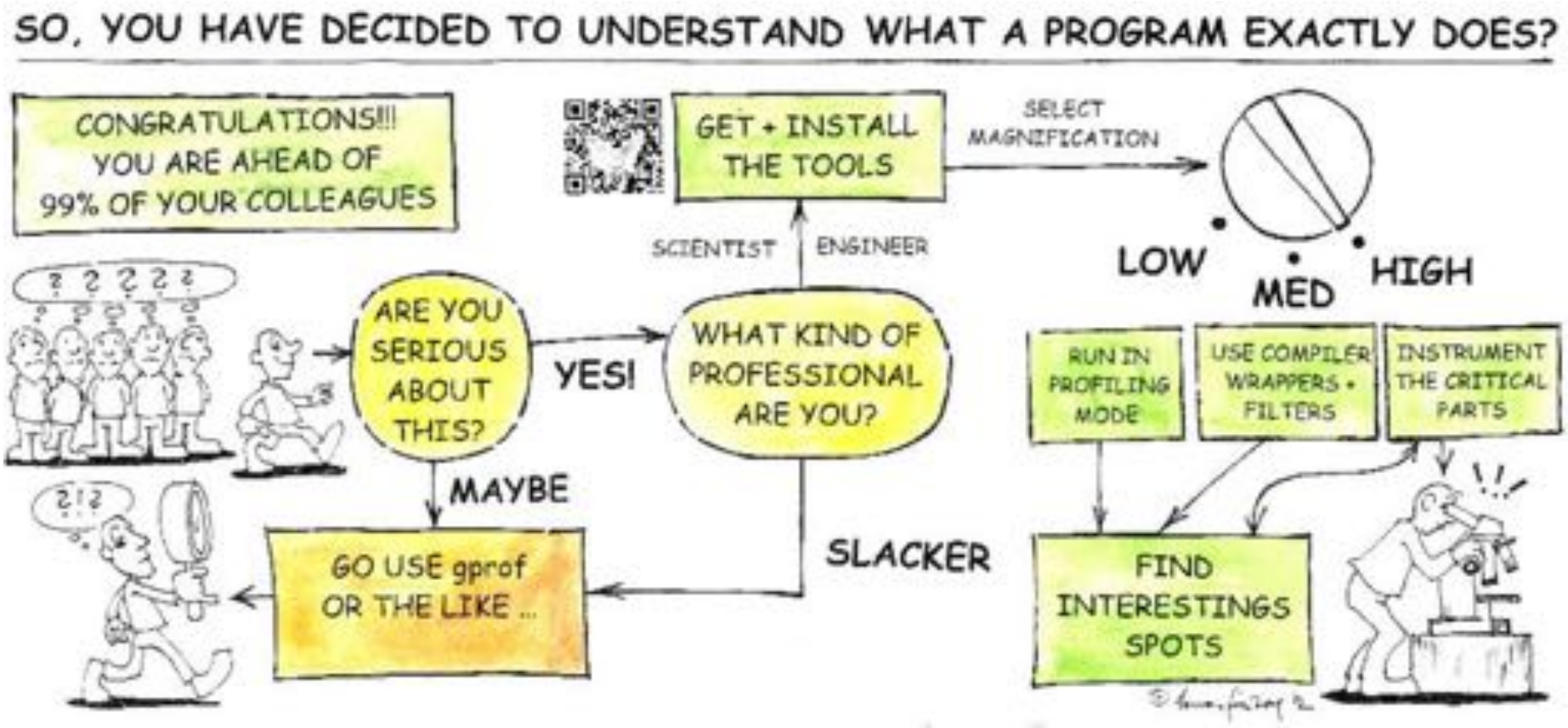
Performance engineering workflow



Performance Analysis Approaches: Profiling vs. Tracing



So what is the right choice?

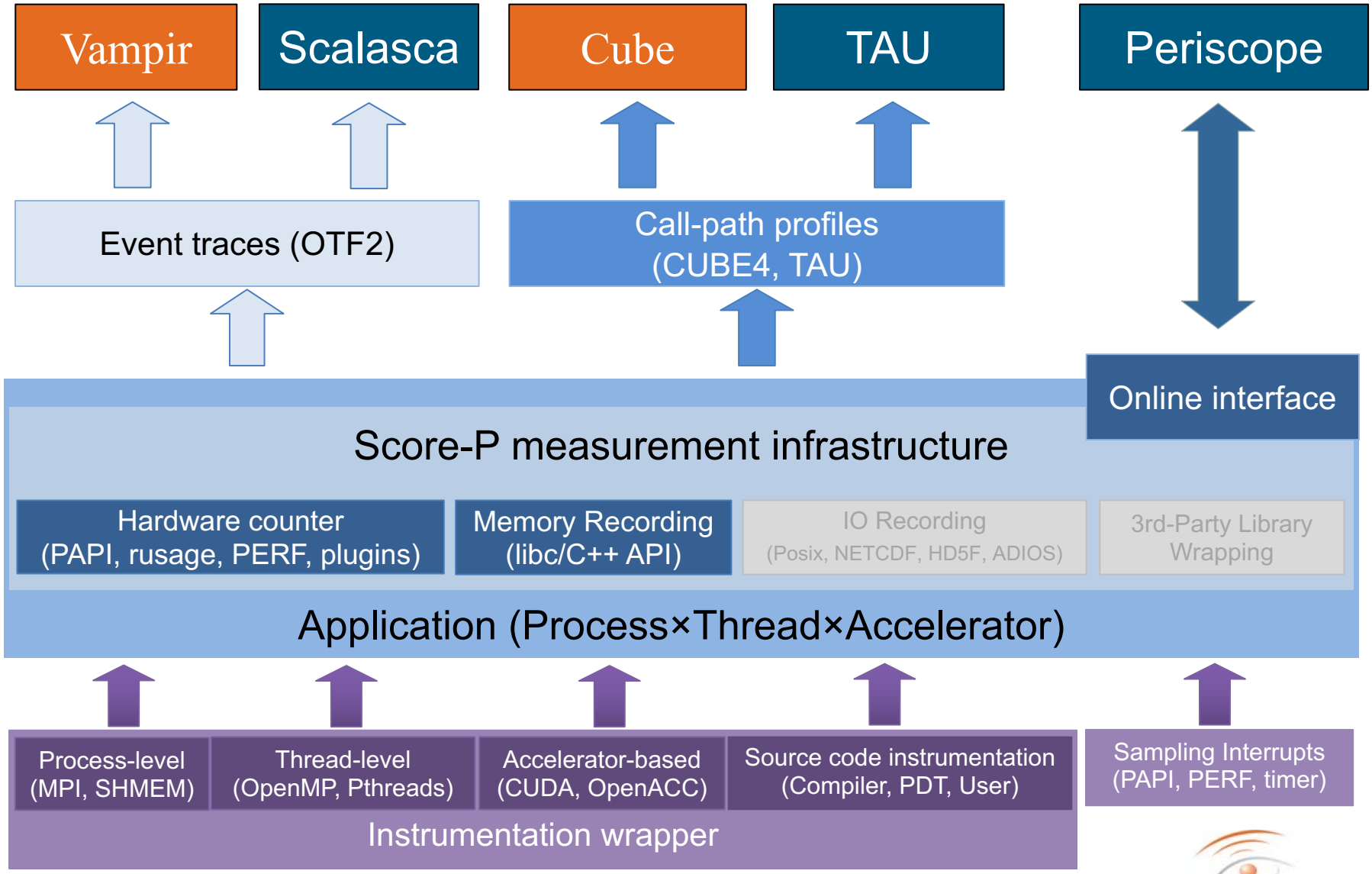


Score-P: Functionality

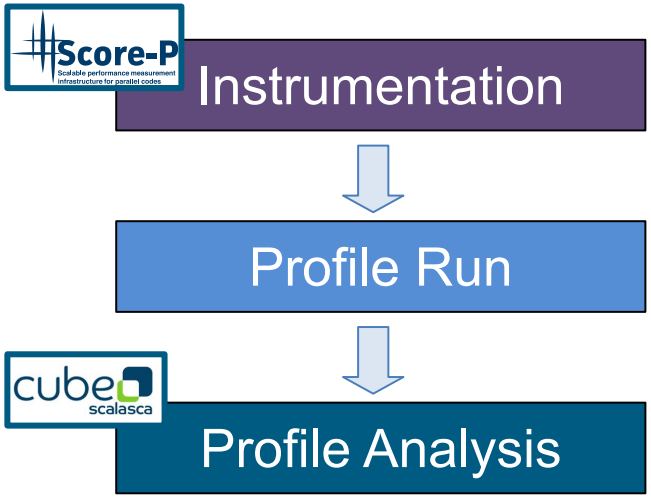
- Typical functionality for HPC performance tools
  - Instrumentation (various methods)
  - Sampling (experimental)
- Flexible measurement without re-compilation
  - Basic and advanced profile generation
  - Event trace recording
- Programming paradigms:
  - Multi-process
    - MPI, SHMEM
  - Thread-parallel
    - OpenMP, Pthreads
  - Accelerator-based
    - CUDA, OpenCL, OpenACC

} Hybrid parallelism

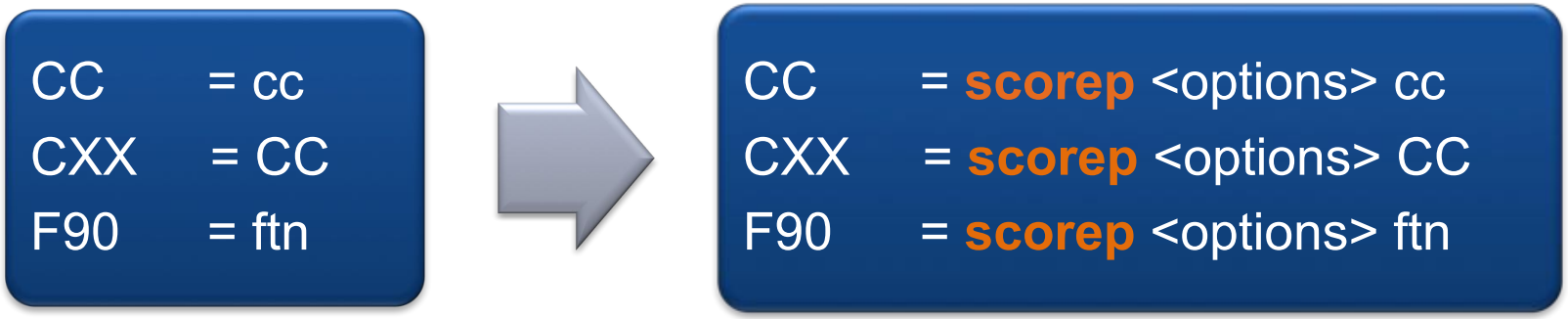
Score-P: Architecture



Score-P Workflow: Profiling



Score-P Workflow: Instrumentation



- To see all available options for instrumentation:

```
$ scorep --help
This is the Score-P instrumentation tool. The usage is:
scorep <options> <original command>

Common options are:
...
  --cuda            Enables cuda instrumentation.
...
  --user            Enables user instrumentation.
...
  --openacc         Enables OpenACC instrumentation.
```

Score-P Workflow: Advanced Instrumentation

- For CMake and autotools based build systems it is recommended to use the scorep-wrapper script instances

Disable instrumentation

```
#CMake

SCOREP_WRAPPER=OFF cmake .. \
   -DCMAKE_C_COMPILER=scorep-icc \
   -DCMAKE_CXX_COMPILER=scorep-icpc \
   -DCMAKE_Fortran_COMPILER=scorep-ifc
```

```
#Autotools

SCOREP_WRAPPER=OFF ../configure \
     CC=scorep-icc \
     CXX=scorep-icpc \
     FC=scorep-ifc \
     --disable-dependency-tracking
```

- Pass instrumentation and compiler flags at make

```
make SCOREP_WRAPPER_INSTRUMENTER_FLAGS="--user" \
     SCOREP_WRAPPER_COMPILER_FLAGS="-g -O2"
```

scorep --user <your_compiler> –g –O2

Score-P Workflow: Measurement

- Measurements are configured via environment variables

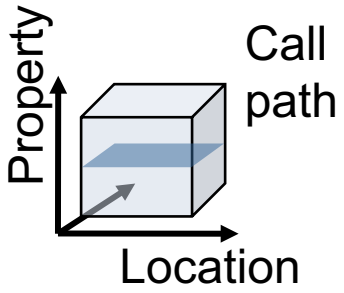
```
$ scorep-info config-vars --full

SCOREP_ENABLE_PROFILING
   [...]
SCOREP_ENABLE_TRACING
   [...]
SCOREP_TOTAL_MEMORY
  Description: Total memory in bytes for the measurement system
   [...]
SCOREP_EXPERIMENT_DIRECTORY
  Description: Name of the experiment directory
   [...]
```

- Example for generating a profile:

```
$ export SCOREP_ENABLE_PROFILING=true
$ export SCOREP_ENABLE_TRACING=false
$ export SCOREP_EXPERIMENT_DIRECTORY=profile

$ aprun <instrumented binary>
```

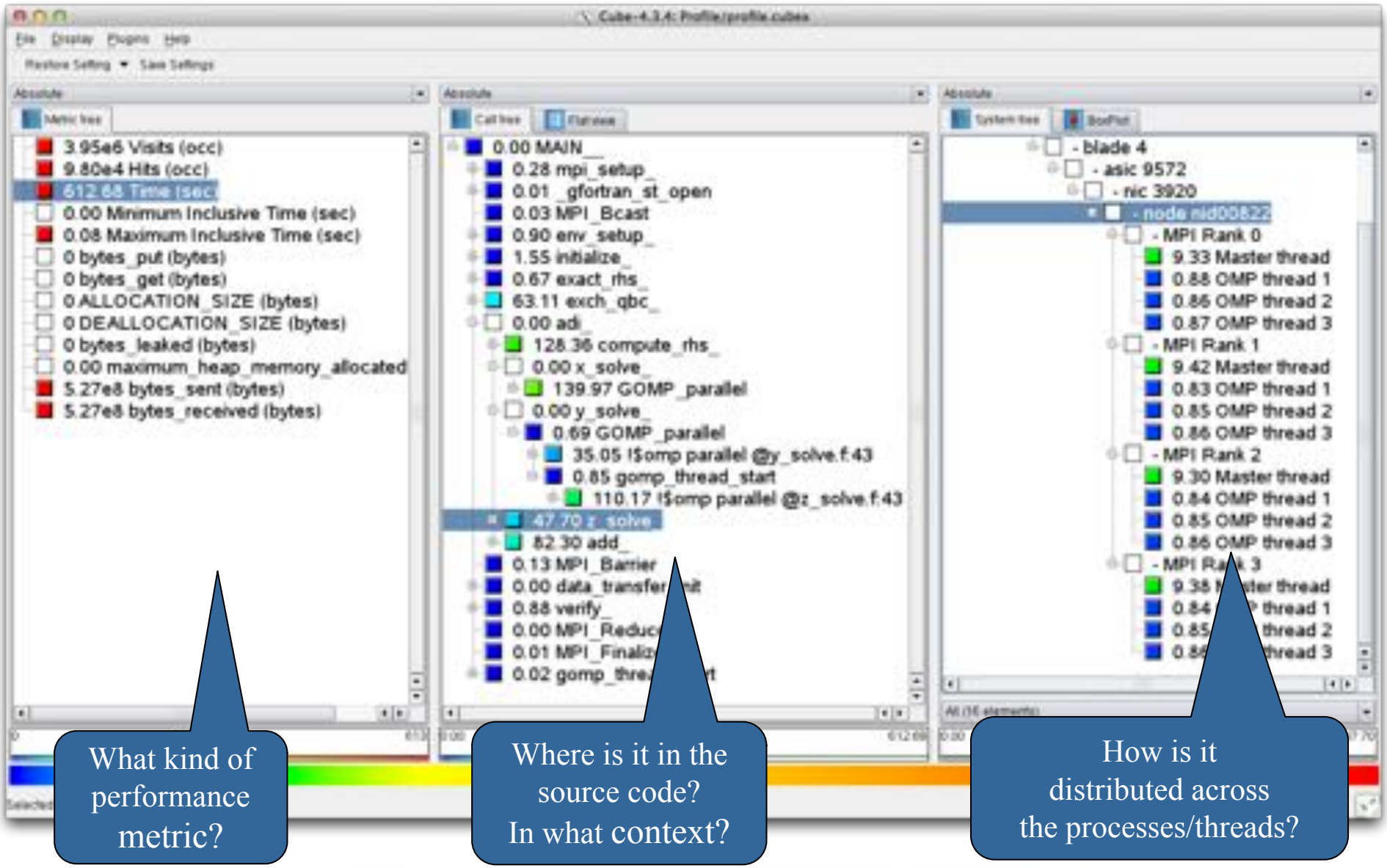
Score-P: Cube

- Profile analysis tool for displaying performance data of parallel programs
- Originally developed as part of Scalasca toolset
- Available as a separate component of Score-P
- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call-tree path (program location)
  - System location (process/thread)
- Three coupled tree browsers

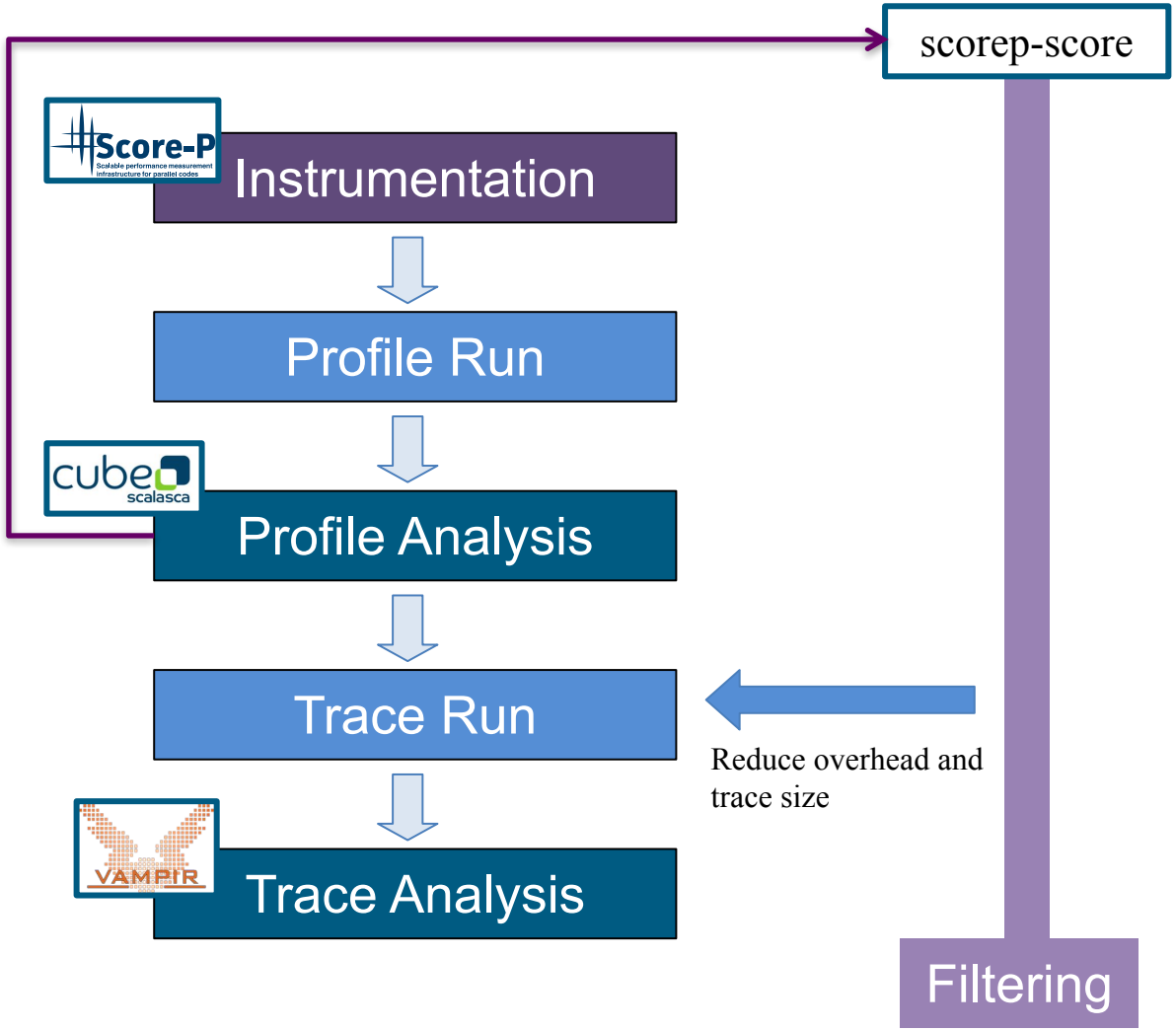


cube scalasca

Score-P: Cube Analysis Presentation



Score-P Workflow: Profiling + Tracing



Score-P Workflow: Filtering

- Use scorep-score to define a filter
  - Exclude short frequently called functions from measurement
    - For profiling: reduce measurement overhead (if necessary)
    - For tracing: reduce measurement overhead and total trace size

```
$ scorep-score -r profile/profile.cubex
Estimated aggregate size of event trace:                   40GB
Estimated requirements for largest trace buffer (max_buf): 10GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       10GB
  [...]
Flt type     max_buf[B]        visits time[s] time[%] time/visit[us]  region
  [...]
     USR  3,421,305,420   522,844,416  144.46    13.4           0.28  matmul_sub
     USR  3,421,305,420   522,844,416  102.40     9.5           0.20  matvec_sub
     USR  3,421,305,420   522,844,416  200.94    18.6           0.38  binvcrhs
     USR    150,937,332    22,692,096    5.58     0.5           0.25  binvrhs
     USR    150,937,332    22,692,096   13.21     1.2           0.58  lhsinit
```

About 10 GB just for these 3 regions per process!

- Filter file:

```
$ vim scorep.filt
SCOREP_REGION_NAMES_BEGIN EXCLUDE
  matmul_sub
  matvec_sub
  binvcrhs
```

Reduce measurement overhead and size of event trace to about 40 GB! (Example uses 4 processes)

Score-P Workflow: Measurement

- Example for generating a trace

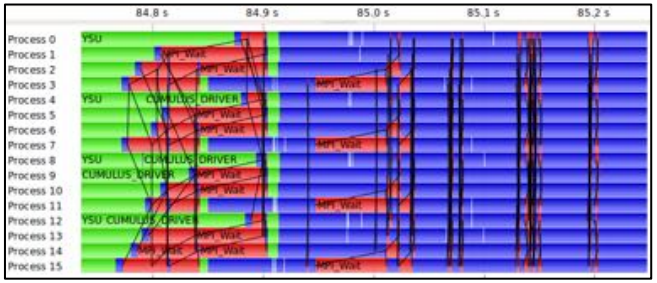
```
$ export SCOREP_ENABLE_PROFILING=false
$ export SCOREP_ENABLE_TRACING=true
$ export SCOREP_EXPERIMENT_DIRECTORY=trace
$ export SCOREP_TOTAL_MEMORY=2G
$ export SCOREP_FILTERING_FILE=scorep.filt

$ aprun <instrumented binary>
```

Vampir: Event Trace Visualization

- Show dynamic run-time behavior graphically at a fine level of detail
- Provide summaries (profiles) on performance metrics

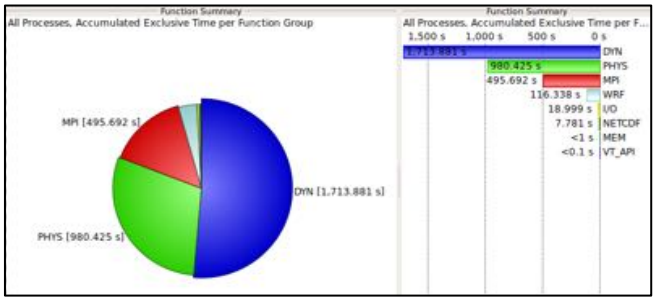
Timeline charts

- Show application activities and communication along a time axis



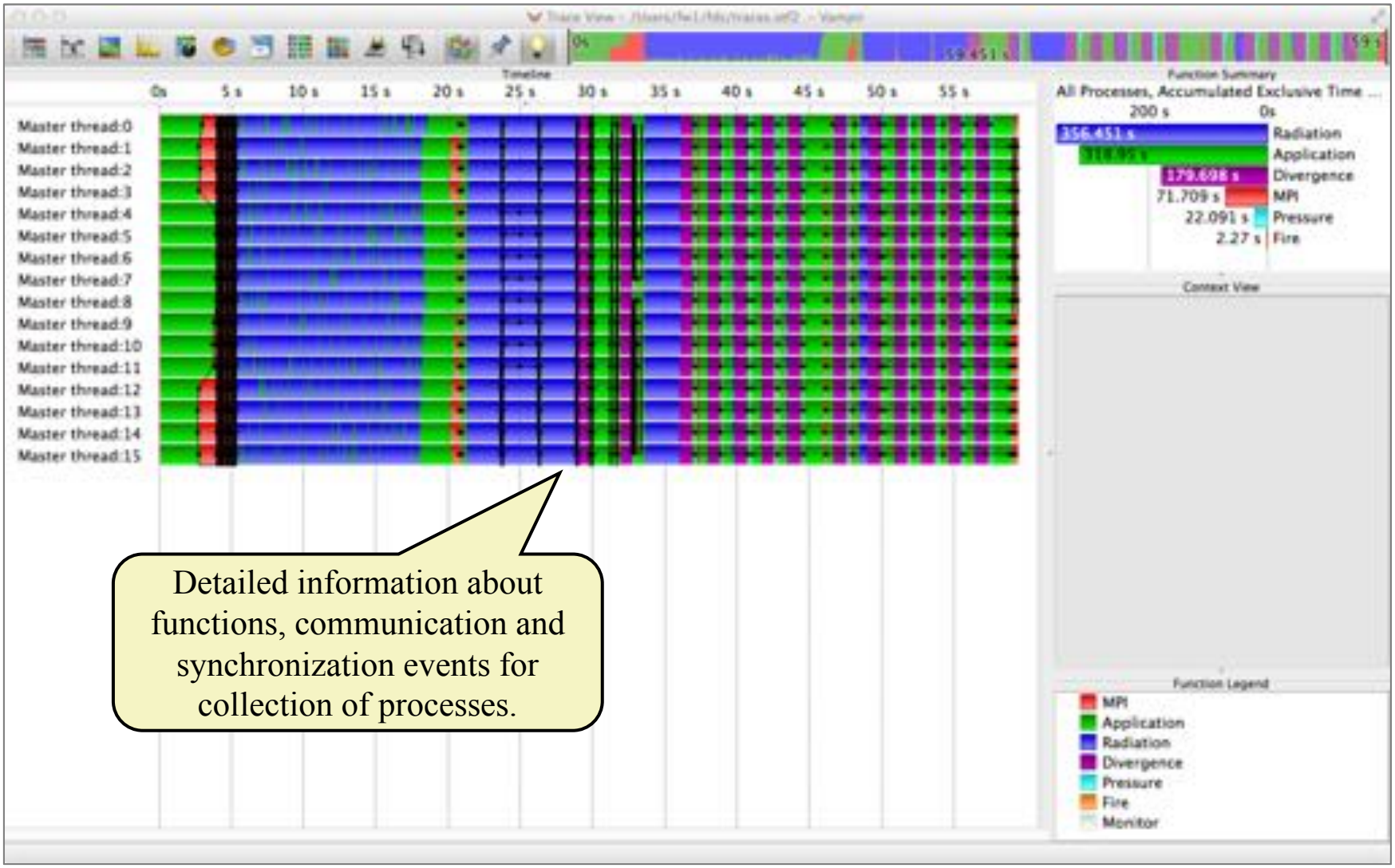
Summary charts

- Provide quantitative results for the currently selected time interval



Vampir: Performance Charts

Master Timeline

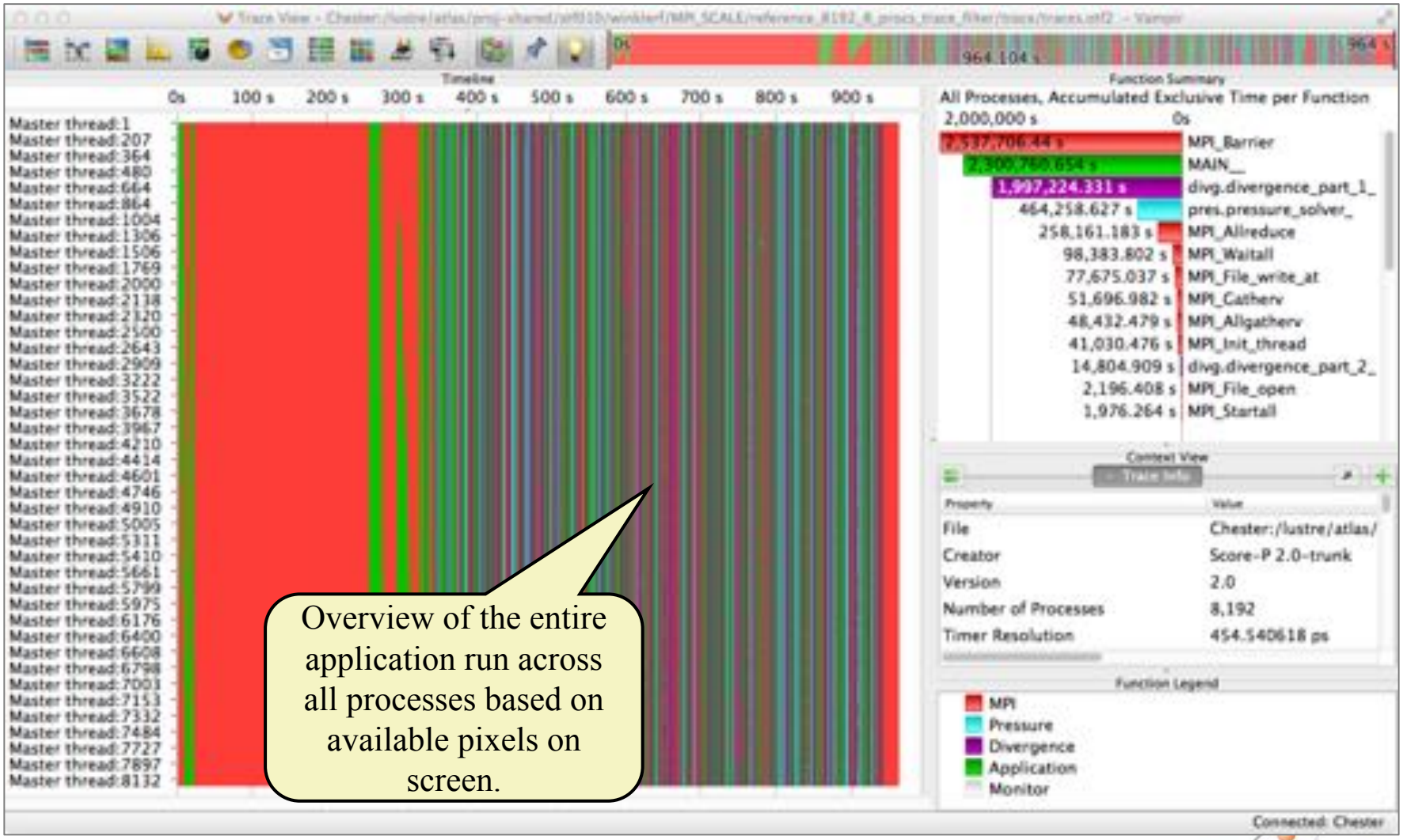


Vampir: Performance Charts

- Trace visualization of FDS (Fire Dynamics Simulator)

Vampir at Scale: FDS with 8192 cores

- Fit to chart height feature in Master Timeline



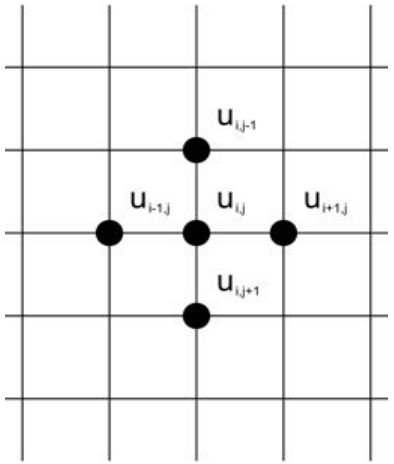
Demo: Jacobi Solver

- Jacobi Example
  - Iterative solver for system of equations

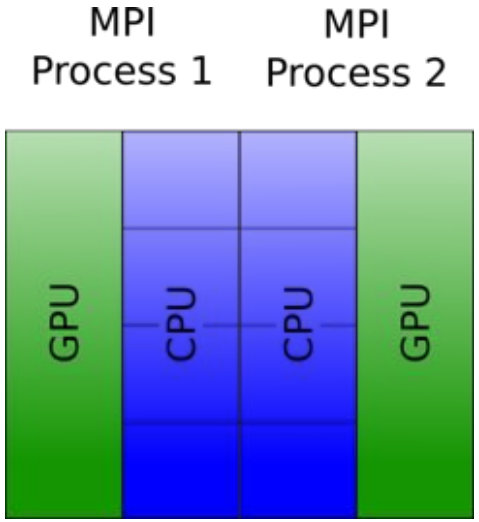
$$U_{old} = U$$

$$u_{i,j} = bu_{old,i,j} + a_x(u_{old,i-1,j} + u_{old,i+1,j}) + a_y(u_{old,i,j-1} + u_{old,i,j+1}) - rHs / b$$

  - Code uses OpenMP, CUDA and MPI for parallelization

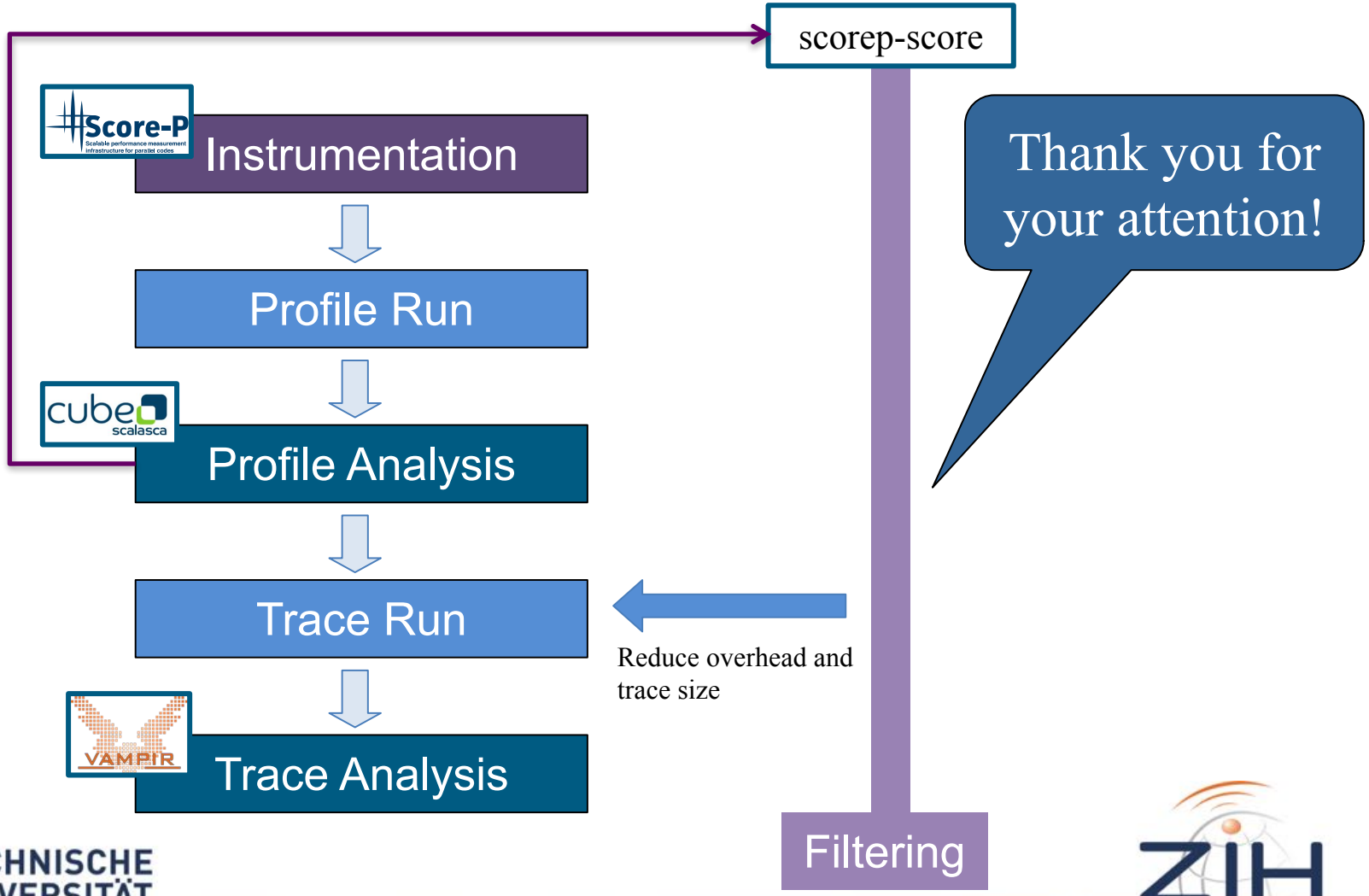


- Domain decomposition
  - Halo exchange at boundaries:
    - Via MPI between processes
    - Via CUDA between hosts and accelerators



The Score-P Tools Infrastructure

- Documentation at https://www.olcf.ornl.gov/support/software/



Demo: Jacobi Solver / Setup

- Connect to Summit-dev and copy sources

```
$ cp /ccs/home/winklerf/scorep_tutorial/jacobi.tar.gz .
$ tar xzvf jacobi.tar.gz
$ cd jacobi
```

- Change programming environment and load modules

```
$ module load gcc/5.4.0
$ module load cuda
$ module load scorep
```

- Compile benchmark and submit job

```
$ make
$ cd bin
$ bsub < run.lsf
$ less run.lsf
Jacobi relaxation Calculation: 8192 x 8192 mesh with
 2 processes and 6 threads + one Tesla P100-SXM2-16GB for each process.
 614 of 4097 local rows are calculated on the CPU to balance the load
 between the CPU and the GPU.
 0, 0.489197
  100, 0.002397
  [...]
 total: 9.409952 s
```

Demo: Jacobi Solver / Profiling

- Build instrumented executable

```
$ make clean
$ make scorep
scorep --cuda cc … -o bin/jacobi_mpi+openmp+cuda
```

- Submit job for profiling run

```
$ less run_profile.lsf
[...]
export SCOREP_ENABLE_PROFILING=true
export SCOREP_ENABLE_TRACING=false
export SCOREP_EXPERIMENT_DIRECTORY=jacobi_profile
export SCOREP_CUDA_ENABLE=yes
[...]
mpirun -n 2 ./jacobi_mpi+openmp+cuda 8192 8192 0.15

$ bsub < run_profile.lsf
$ less jacobi.o[JOB_ID]
Jacobi relaxation Calculation: 8192 x 8192 mesh with
 2 processes and 6 threads + one Tesla P100-SXM2-16GB for each process.
[...]
 total: 10.678350 s
```

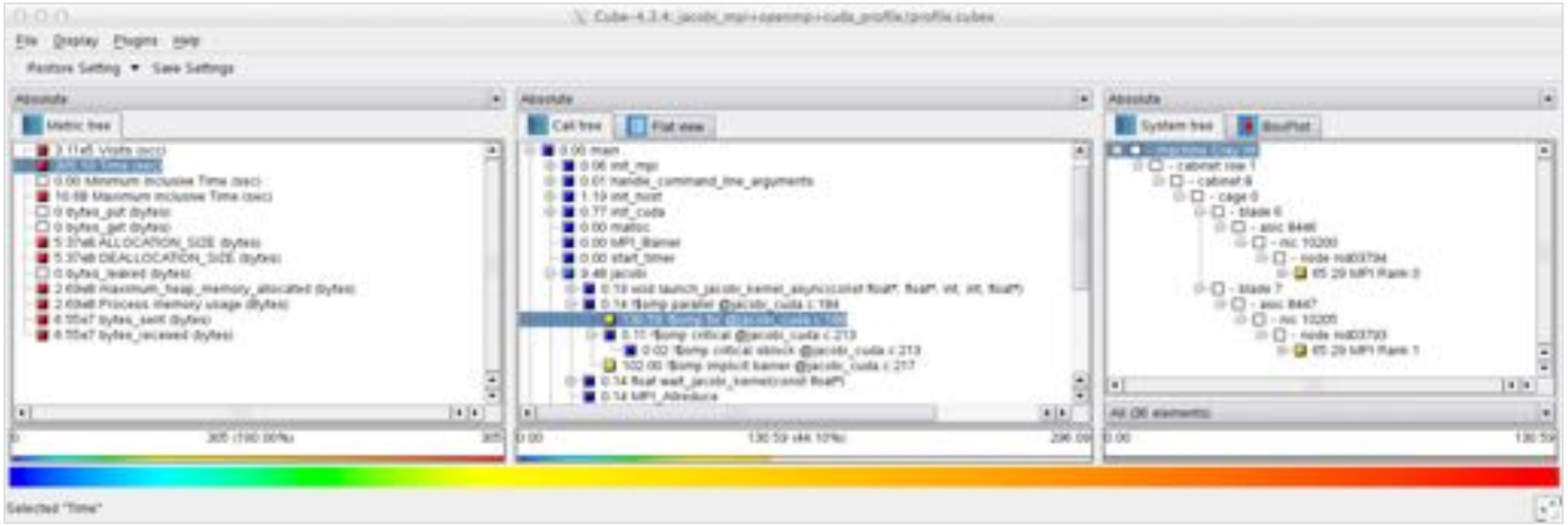
Demo: Jacobi Solver / Profile Analysis

- Perform flat profile analysis with cube_stat

```
$ cd bin
$ cube_stat -t 10 -p jacobi_mpi+openmp+cuda_profile/profile.cubex
cube::Region                        NumberOfCalls ExclusiveTime InclusiveTime
!$omp for @jacobi_cuda.c:188     32000.000000    131.797289    131.797289
!$omp implicit barrier           32000.000000    104.298683    104.298683
!$omp for @jacobi_cuda.c:258     32000.000000     42.999056     50.568642
[...]
```

- Perform call-path profile analysis with Cube

```
$ cube jacobi_profile/profile.cubex
```



Demo: Jacobi Solver / Scoring

- Do we need a filter? (Overhead and memory footprint)

```
$ scorep-score jacobi_profile/profile.cubex
Estimated aggregate size of event trace:                   10MB
Estimated requirements for largest trace buffer (max_buf): 5MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       41MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=41MB to avoid intermediate
 flushes or reduce requirements using USR regions filters.)

flt     type max_buf[B]  visits time[s] time[%] time/visit[us]  region
         ALL  4,924,060 310,504  308.53   100.0         993.63  ALL
         OMP  4,135,850 256,417  287.31    93.1        1120.46  OMP
        CUDA    494,338  38,025   10.40     3.4         273.53  CUDA
         COM    156,260  12,020   10.46     3.4         870.58  COM
         MPI    137,222   4,012    0.30     0.1          73.96  MPI
      MEMORY        260      20    0.06     0.0        2972.15  MEMORY
         USR        130      10    0.00     0.0          10.26  USR
```

No filtering required.

Demo: Jacobi Solver / Tracing

- Submit job for tracing run

```
$ cd ..
$ less run_trace.lsf
[...]
export SCOREP_ENABLE_PROFILING=false
export SCOREP_ENABLE_TRACING=true
export SCOREP_EXPERIMENT_DIRECTORY=jacobi_trace
export SCOREP_CUDA_ENABLE=yes
export SCOREP_TOTAL_MEMORY=50MB
[...]
mpirun -n 2 ./jacobi_mpi+openmp+cuda 8192 8192 0.15

$ bsub < run_trace.lsf
$ less jacobi.o[JOB_ID]
Jacobi relaxation Calculation: 8192 x 8192 mesh with
 2 processes and 6 threads + one Tesla P100-SXM2-16GB for each process.
 614 of 4097 local rows are calculated on the CPU to balance the load
 between the CPU and the GPU.
    0, 0.489197
  100, 0.002397
  [...]
  900, 0.000269
 total: 9.895828 s
```

Demo: Jacobi Solver / Trace Analysis

- Perform analysis on the trace data with Vampir

```
$ cd bin
$ module load vampir
$ vampir jacobi_trace/traces.otf2
```