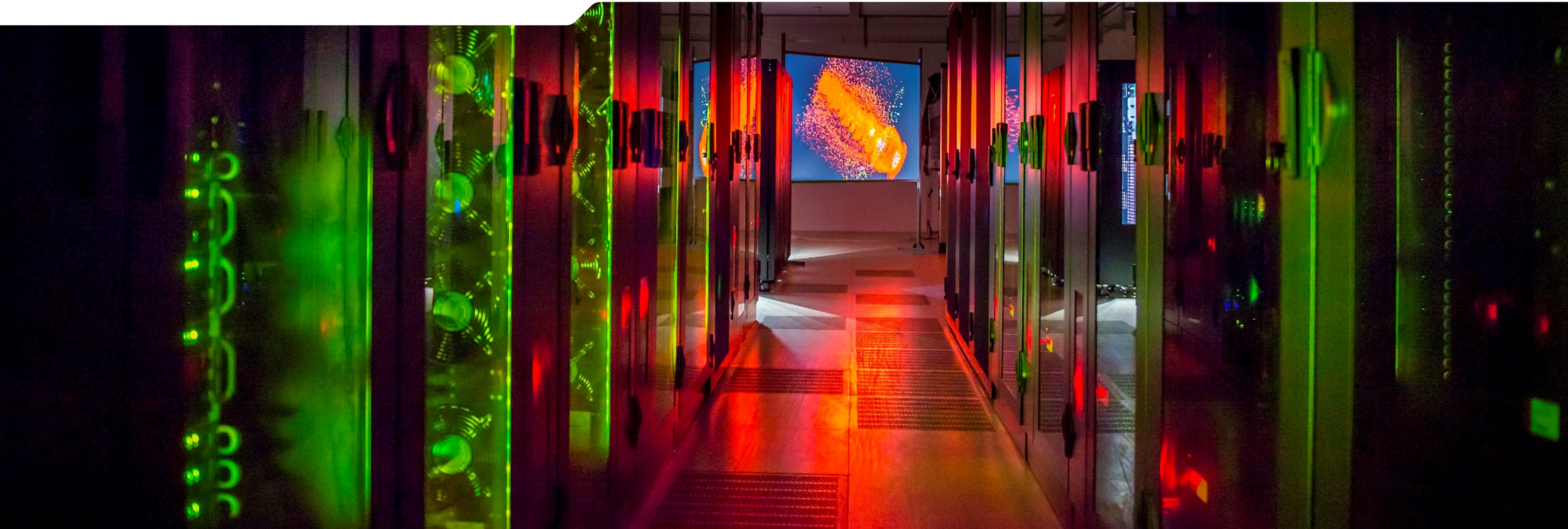
S9347: Performance Analysis for Large Scale GPU Applications and DL Frameworks

HZDR
HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF



Dr. Guido Juckeland / Robert Henschel

Head Computational Science Dept. / Director of Science Community Tools at Indiana University

Agenda

What to expect from the next 80 minutes

- Motivation
- Generating profiles and trace files with Score-P
- Visualizing trace files with Vampir
- Looking into Deep Learning Frameworks

Disclaimer

It‘s extremely easy to waste performance

- Poor/no GPU usage (80-90%)
- Bad MPI (50-90%)

- **Total: 1% of peak (or worse)**

- **Performance tools will not “automagically” make your code faster – they just point to “areas of interest”**

Motivation

Performance Tuning 101

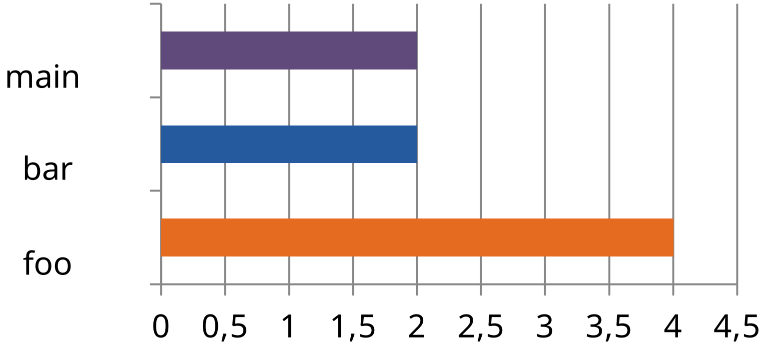
Profiling vs. Tracing

Preserving the details

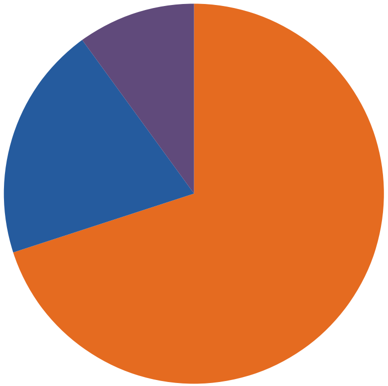
GPU CENTER OF EXCELLENCE

Statistics

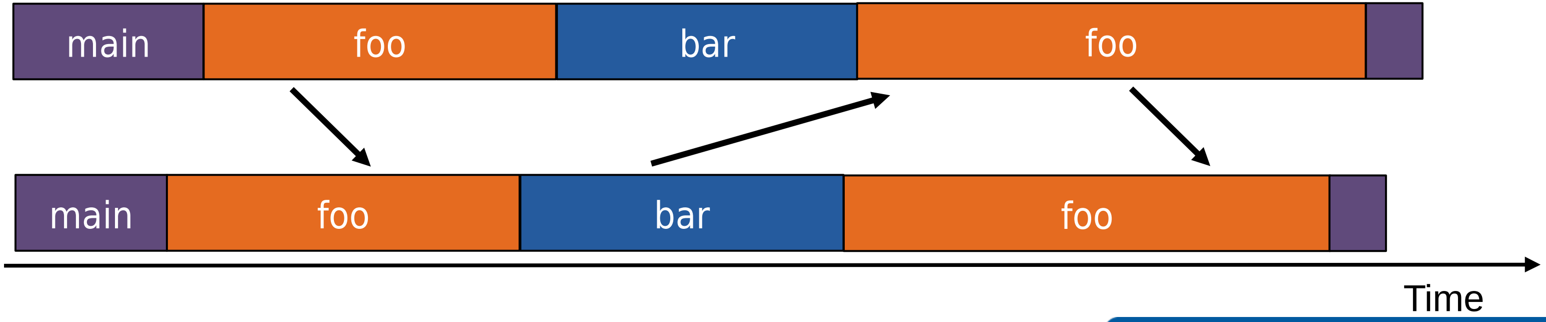
Number of Invocations



Execution Time

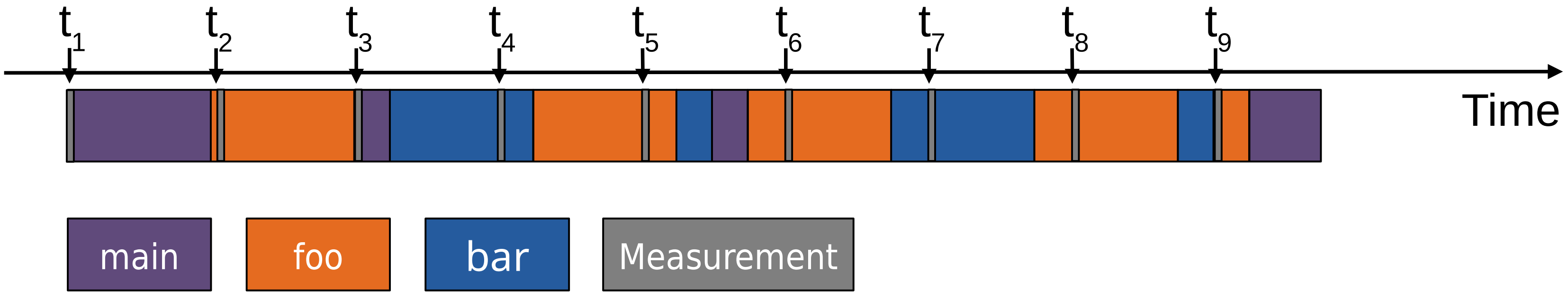


Timelines



Sampling

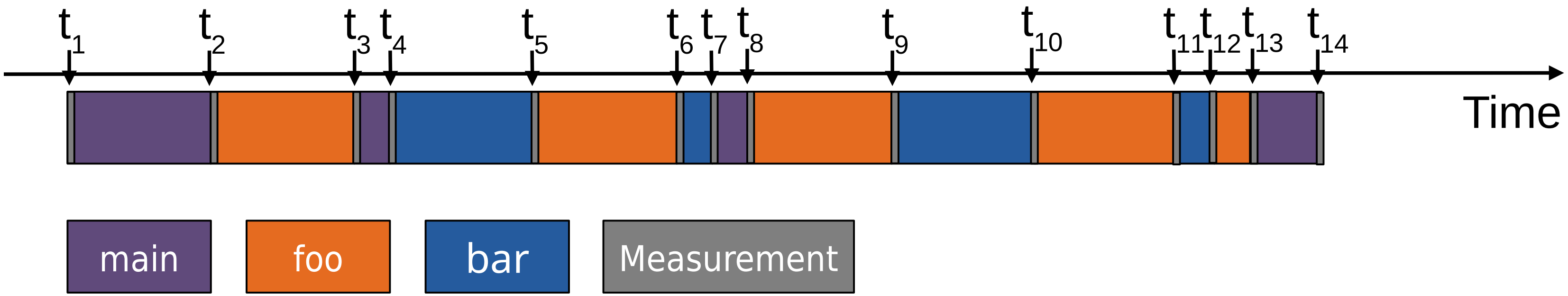
Periodic observations of your application (Pull)



- Running program is periodically interrupted to take measurement
- Statistical inference of program behavior
  - Not very detailed information on highly volatile metrics
  - Requires long-running applications
- Works with unmodified executables

Instrumentation

Modify application to deliver information (Push)



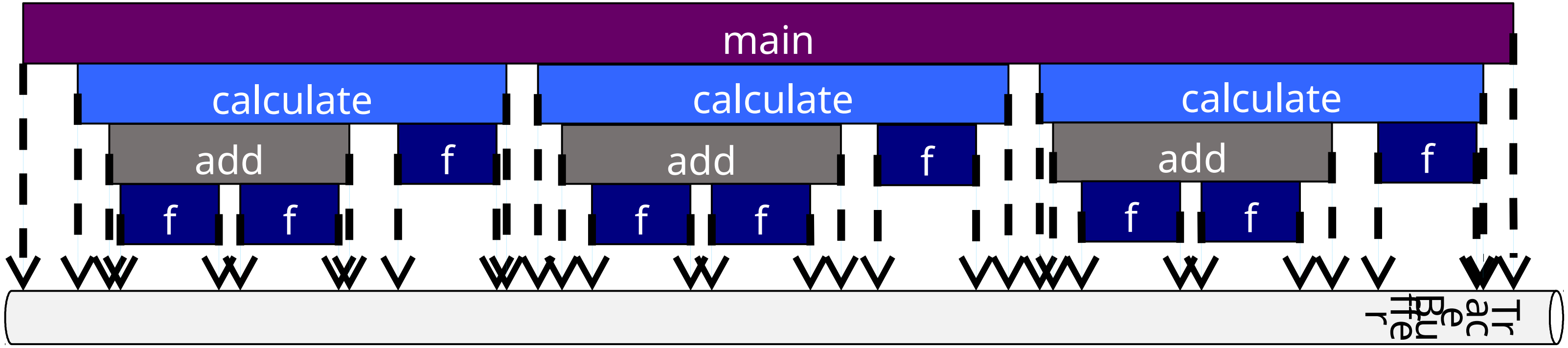
- Measurement code is inserted such that every event of interest is captured directly
- Advantage:
  - Much more detailed information
- Disadvantage:
  - Processing of source-code / executable necessary
  - Large relative overheads for small functions

Sampling vs. Tracing

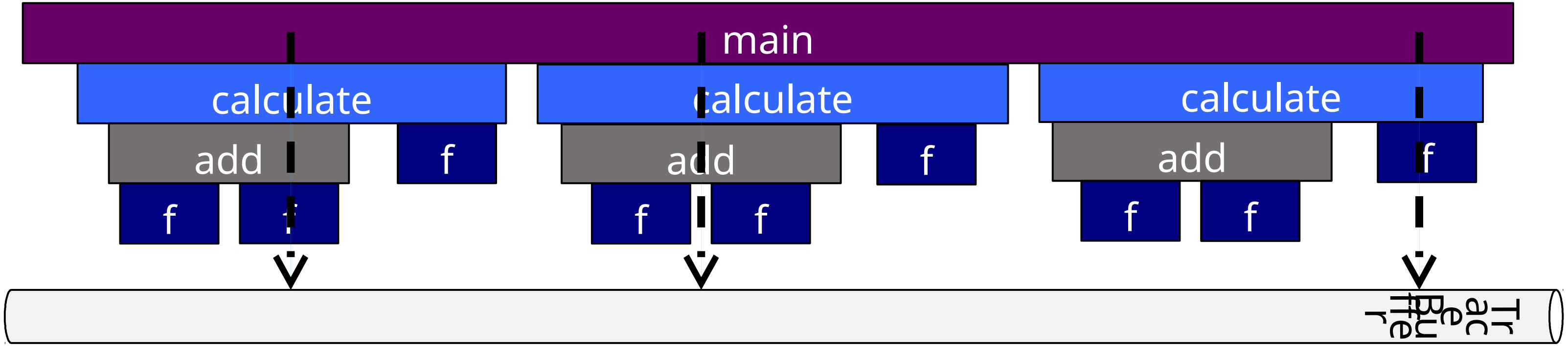
Comparing both approaches visually

NVIDIA GPU CENTER OF EXCELLENCE

Function Instrumentation:



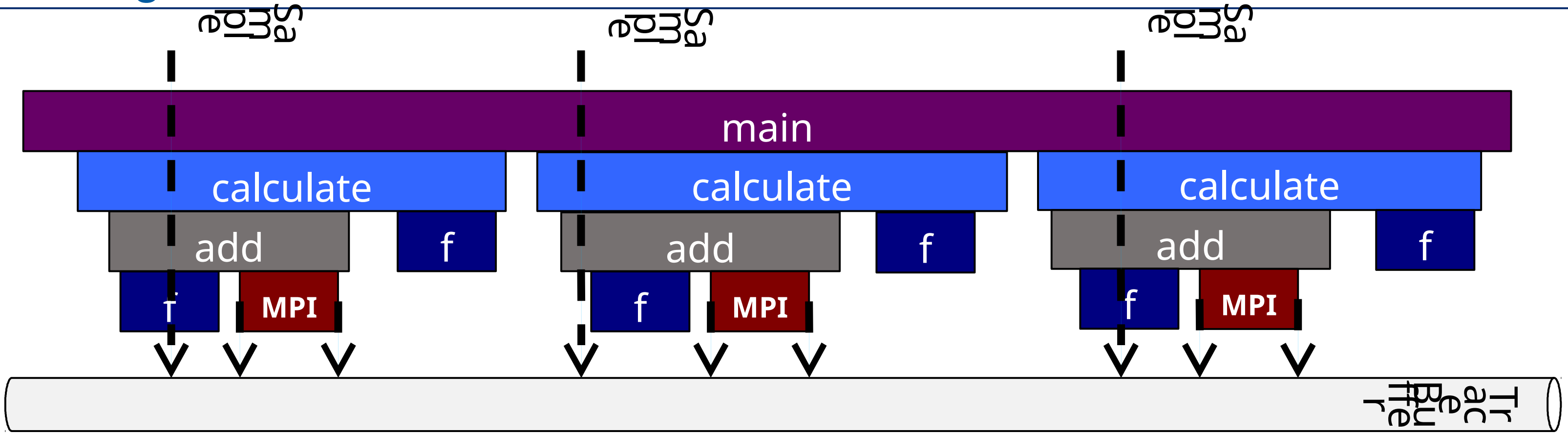
Sampling:



Sampling + Instrumentation

Combining the best of both worlds

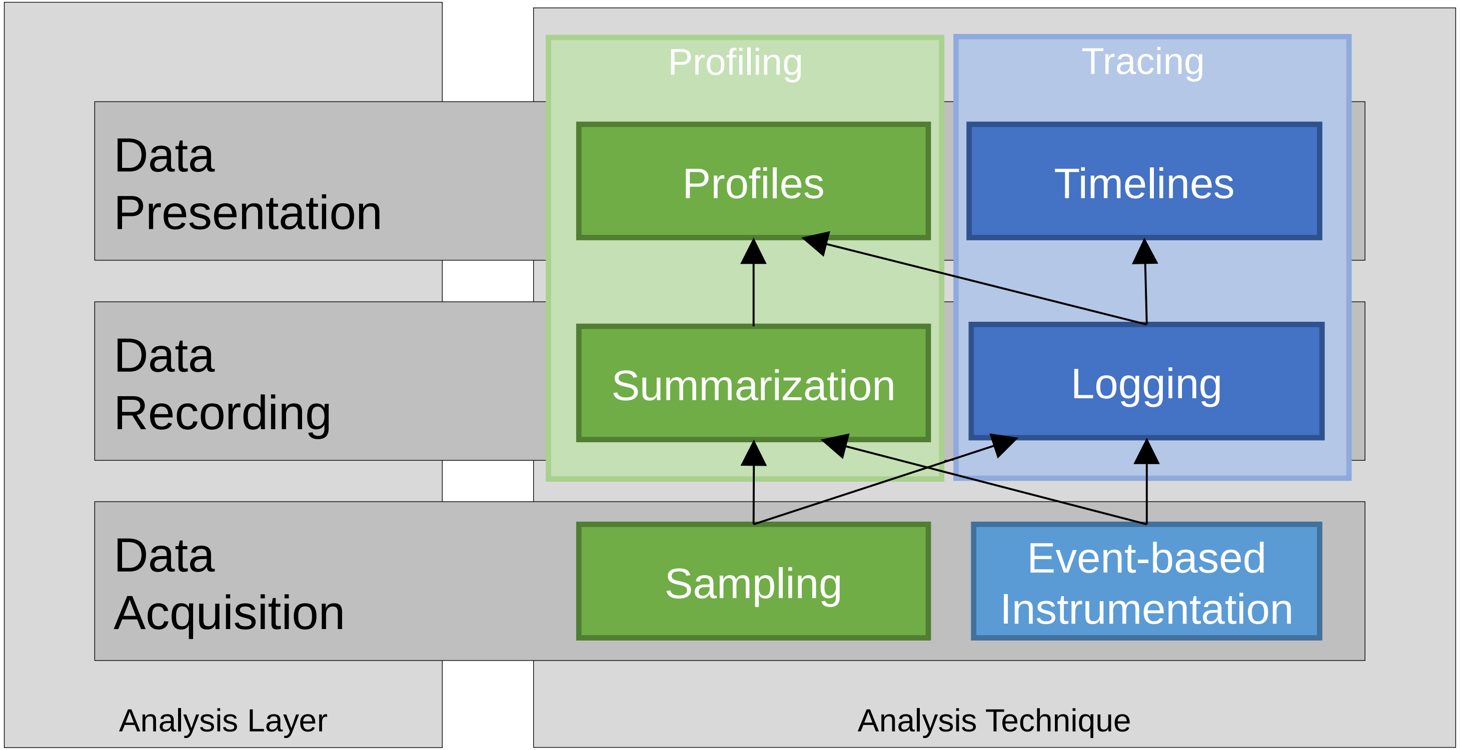
NVIDIA GPU CENTER OF EXCELLENCE



- Long running applications:
  - Requires large buffers or heavy filtering
  - Creating a filter requires runs in advance
- Codes with many small functions (e.g.: C++):
  - Function instrumentation a challenge
- **Score-P: Sampling+Tracing**

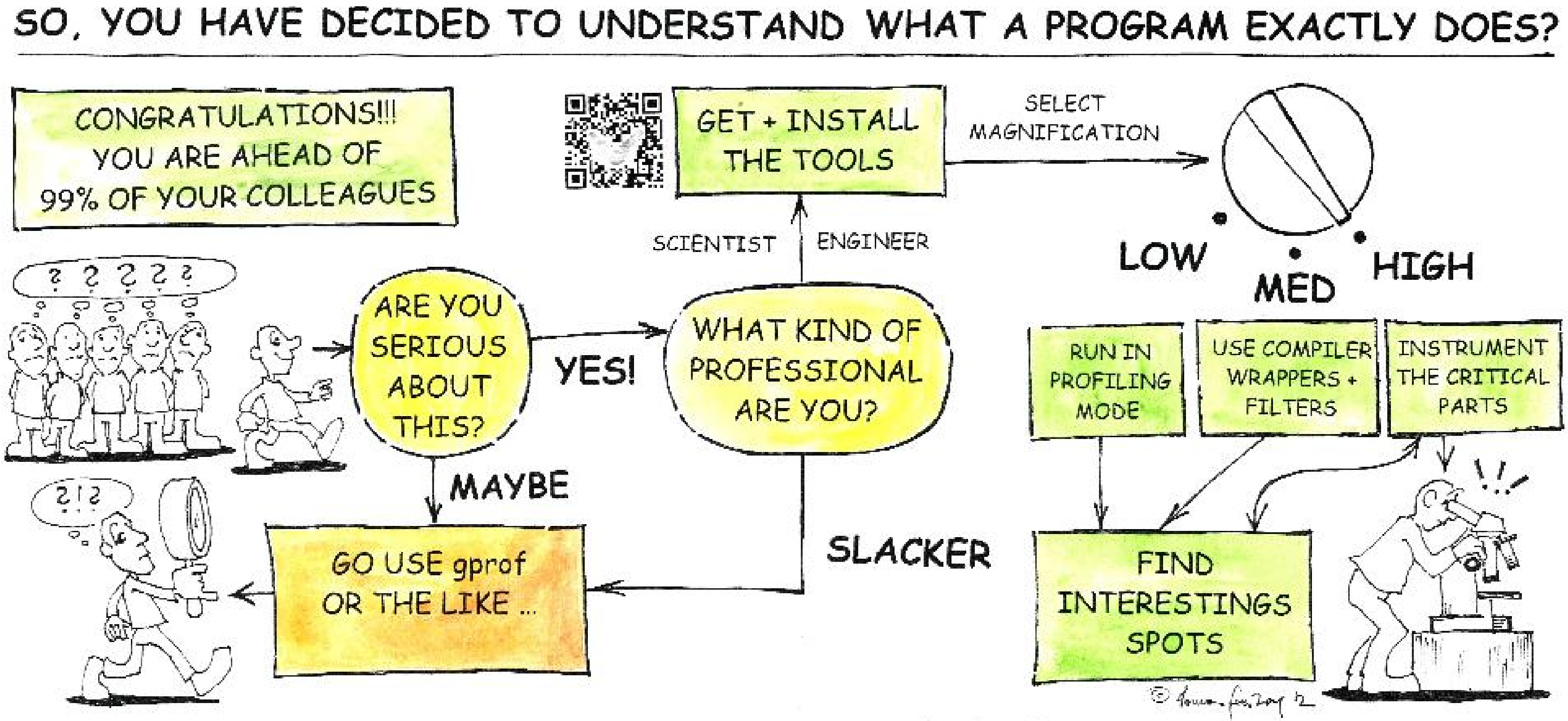
Terms and How They Relate

Making sure we use the same words



Summary

Making the “right” choices

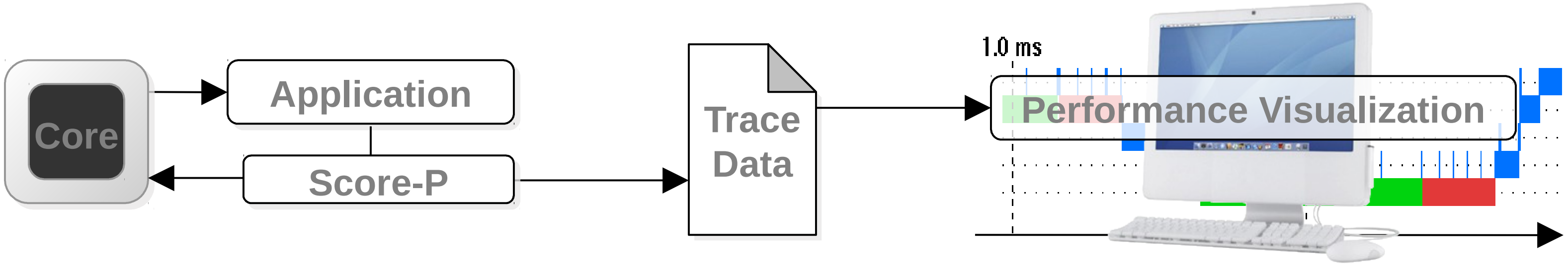


Generating Traces and Profiles

with Score-P

Overall workflow

Recording and studying performance data



- Attach Score-P to application
- Run with attached monitor ==> trace/profile data
- Study trace with Vampir / profile with Cube
- Repeat to:
  - Adapt instrumentation (“what you measure”)
  - Evaluate result of a change

Attaching Score-P

a.k.a. instrumenting your source code

```
CC    = pgcc
CXX   = pgCC
F90   = pgf90
MPICC = mpicc
NVCC  = nvcc
```

```
CC    = scorep <options> pgcc
CXX   = scorep <options> pgCC
F90   = scorep <options> pgf90
MPICC = scorep <options> mpicc
NVCC  = scorep <options> nvcc
```

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

Common options are:
...
  --instrument-filter=<file>
                  Specifies the filter file for filtering functions during
                  compile-time. It applies the same syntax, as the one
                  used by Score-P during run-time.

  --user          Enables user instrumentation.
```

Attaching Score-P

Instrument once – change measurement via runtime 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
    [...]
```

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

$ mpirun <instrumented binary>
```

Profiling Example

Combined Sampling+Tracing

Available since Score-P 2.0

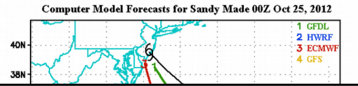
NVIDIA GPU CENTER OF EXCELLENCE

```
$ export SCOREP_ENABLE_TRACING=true
$ export SCOREP_ENABLE_UNWINDING=true
$ export SCOREP_SAMPLING_EVENTS=perf_cycles@2000000
```

- User code is sampled (pull)
- Runtime libraries with tracing support use events (push):
  - MPI
  - OpenMP / OpenACC / pthreads
  - CUDA / OpenCL
  - I/O

Things to look at

What can Score-P record?



User Functions

- C/C++/Fortran
- Sampling *NEW*
- Custom regions

- Java
- Python (*Experimenal*)

Parallel Paradigms

- MPI
- Pthreads
- OpenMP
- XeonPhi Native *NEW*
- CUDA
- OpenACC/OpenCL *NEW*
- OpenShmem (+Cray)
- I/O (*Experimental*)

Hardware

- Performance counters (PAPI)
- Plugin counters

Operating System

- Resource usage

GPU Tracing

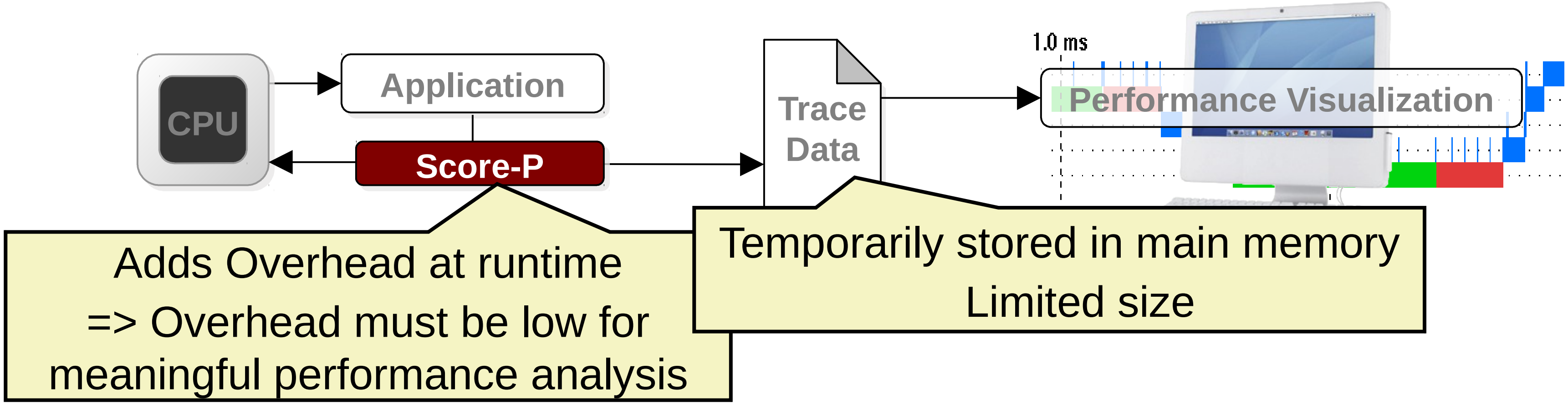
Example CUDA and OpenACC

```
$ export SCOREP_ENABLE_TRACING=yes
$ export SCOREP_TIMER=clock_gettime
$ export SCOREP_CUDA_ENABLE=driver,kernel,memcpy,flushatexit
$ export SCOREP_OPENACC_ENABLE=yes
$ export ACC_PROFLIB=$SCOREP_LIB/libscorep_adapter_openacc_event.so
```

- Can be used in combination
- Also supports CUPTI counters

Limitations

Why tracing is hard



- Event tracing requires trade-offs:
  - Only add the data sources you need
  - Limit granularity (i.e., filtering)
- Score-P is a profiling experiment

DEMO:

Generating Traces and Profiles with Score-P

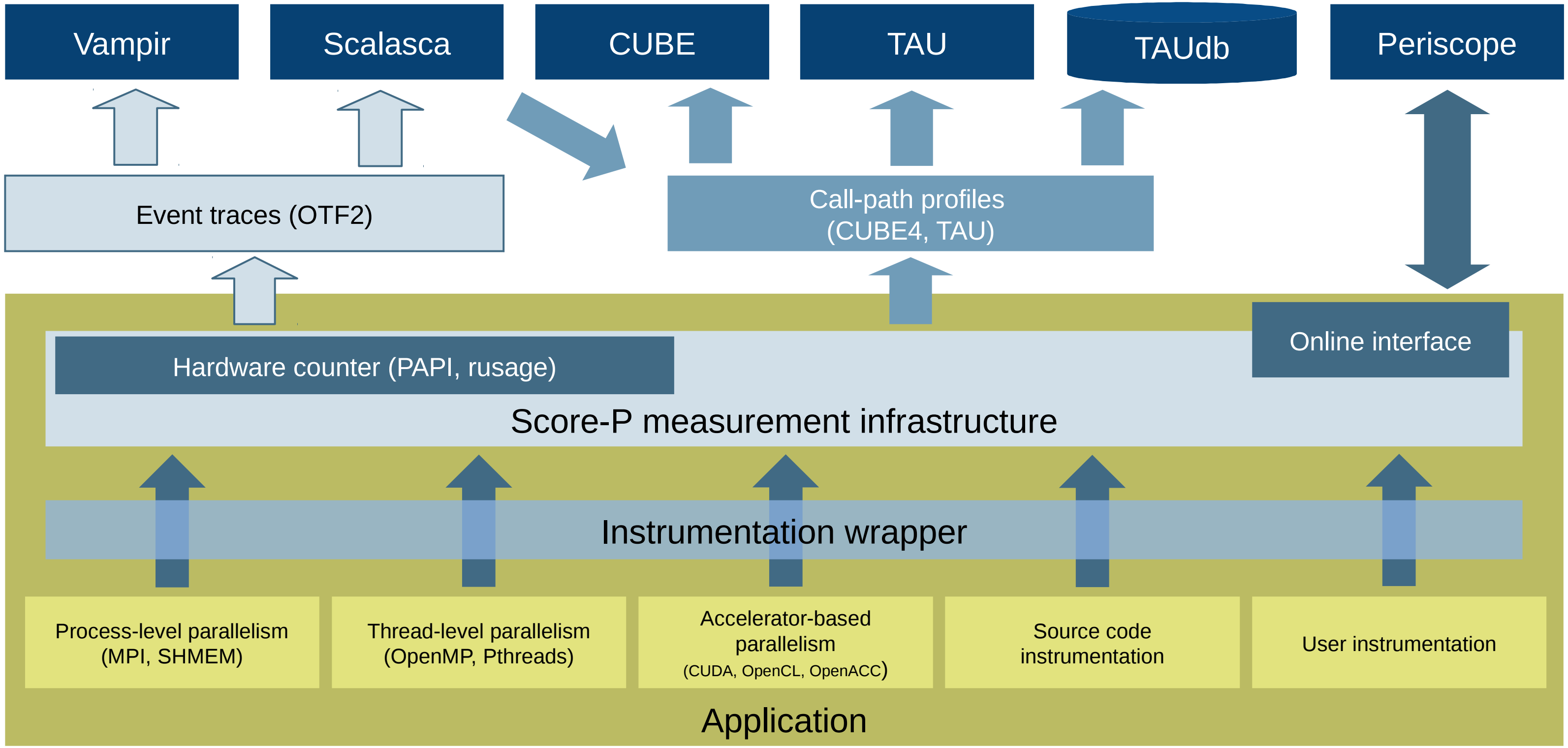
Visualizing

Profiles with CUBE

Traces with Vampir

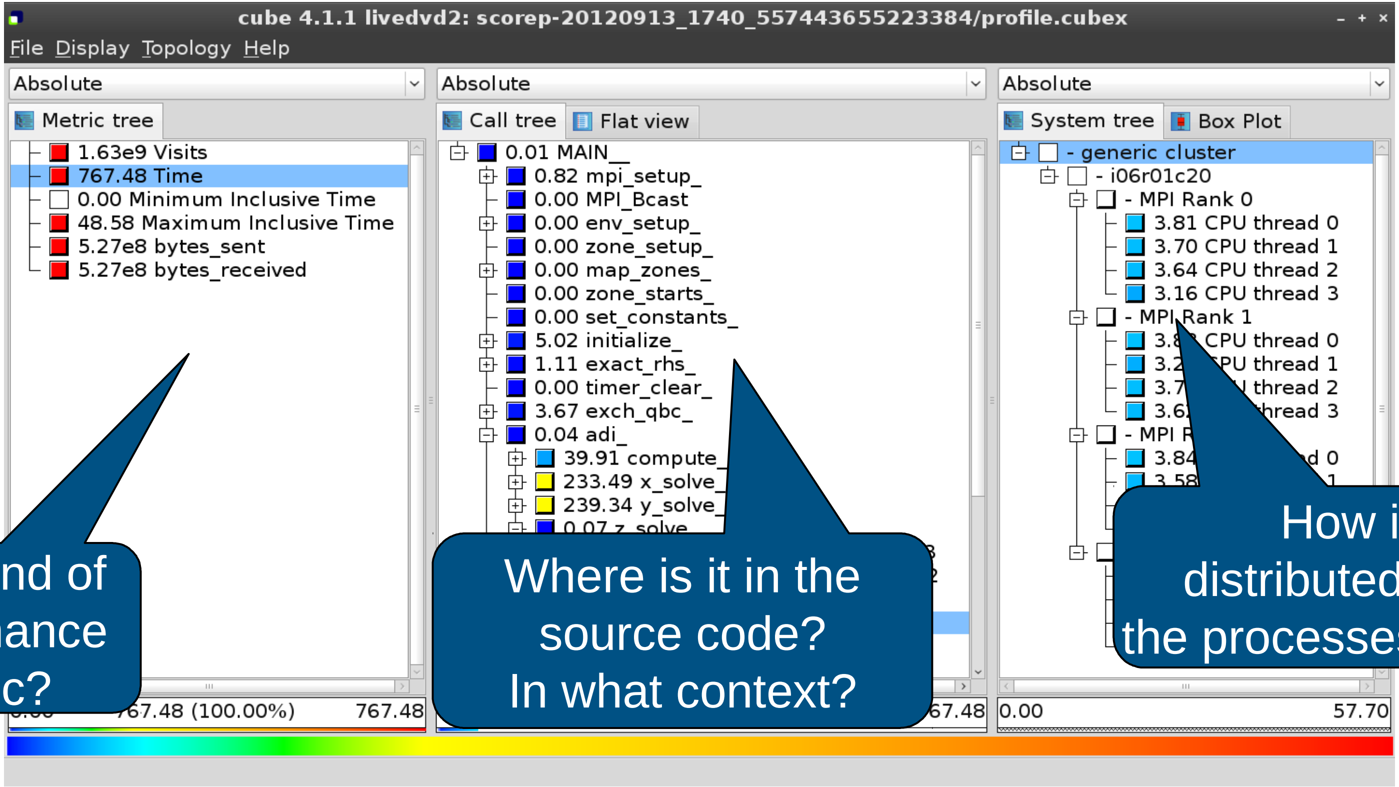
Bringing it all together

Score-P + Analysis Tools



CUBE

Interactive profile analysis



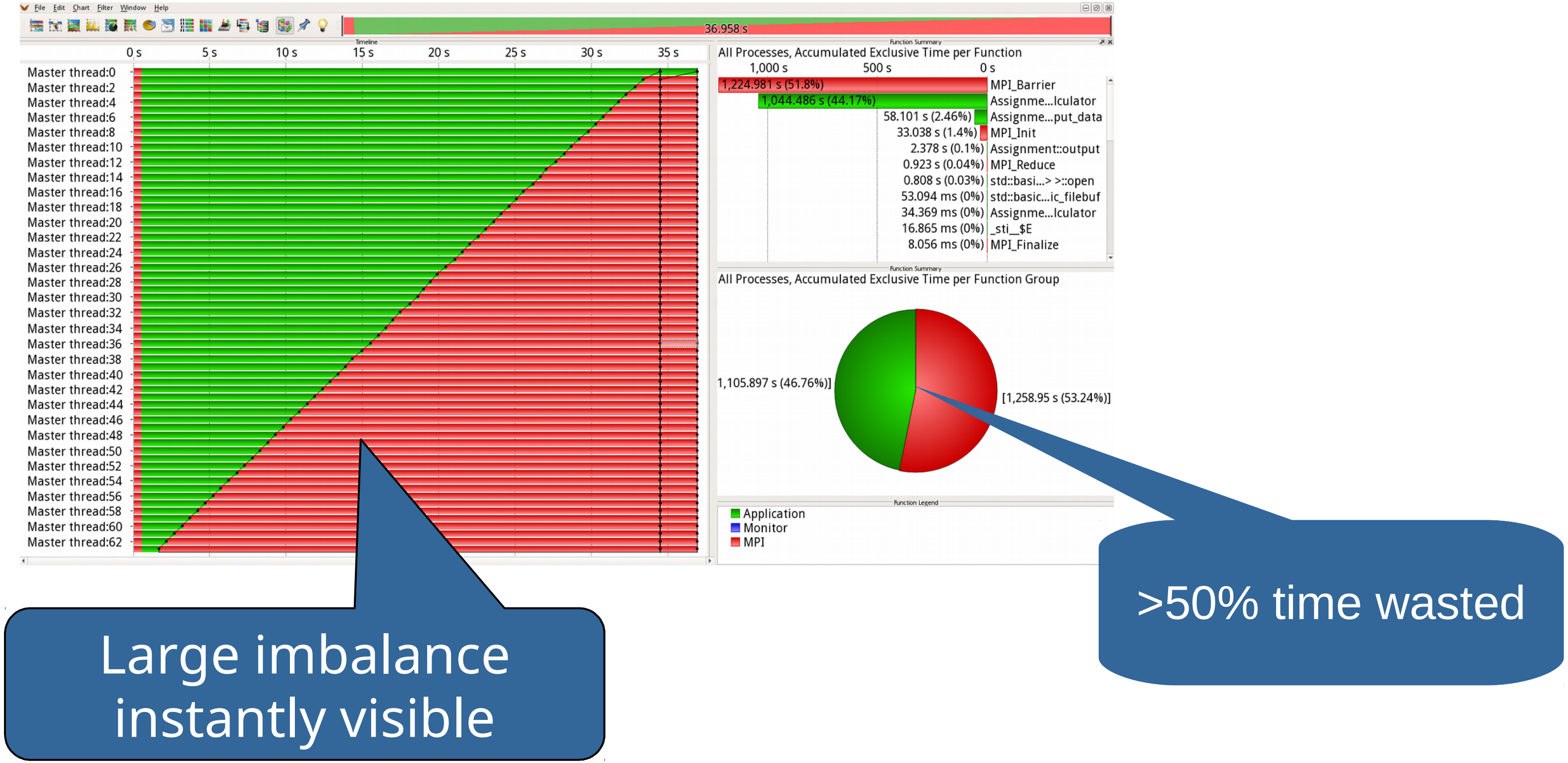
What kind of performance metric?

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

How is it distributed across the processes/threads?

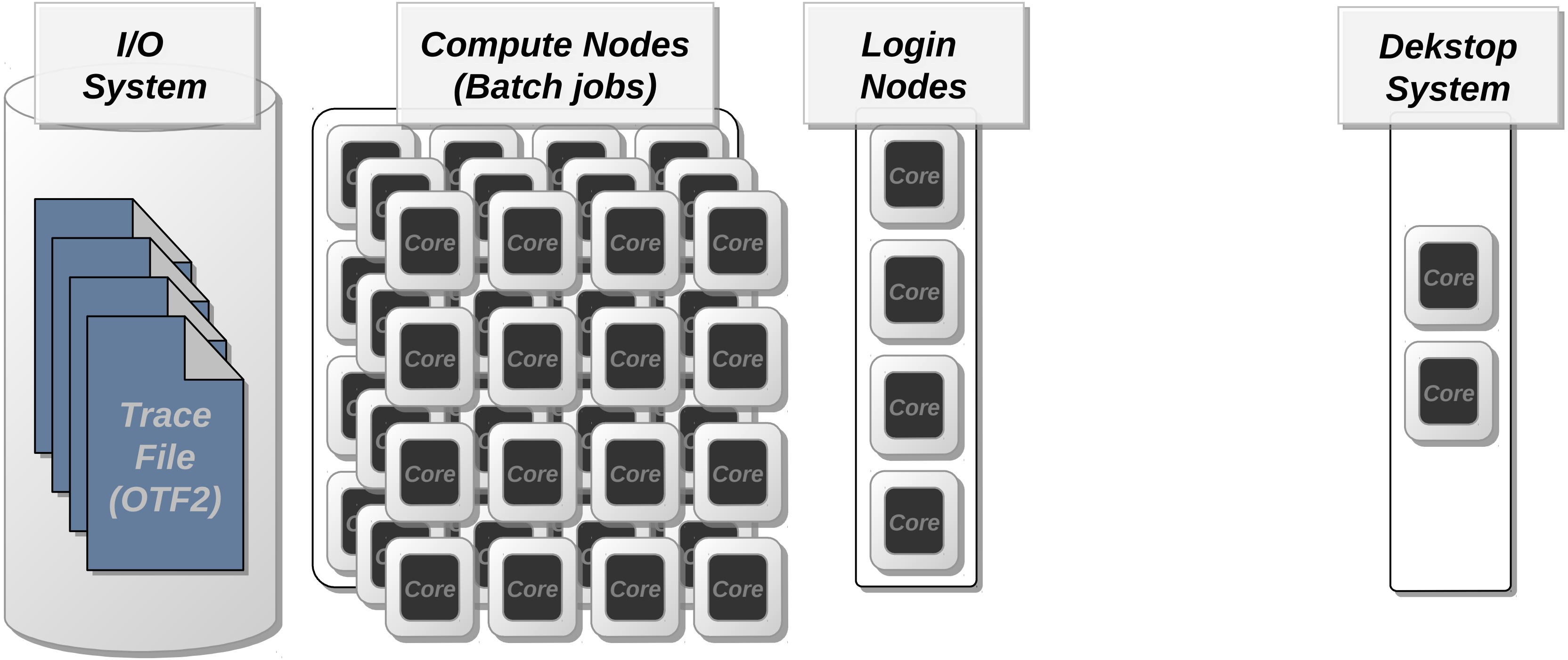
Vampir

Interactive trace analysis



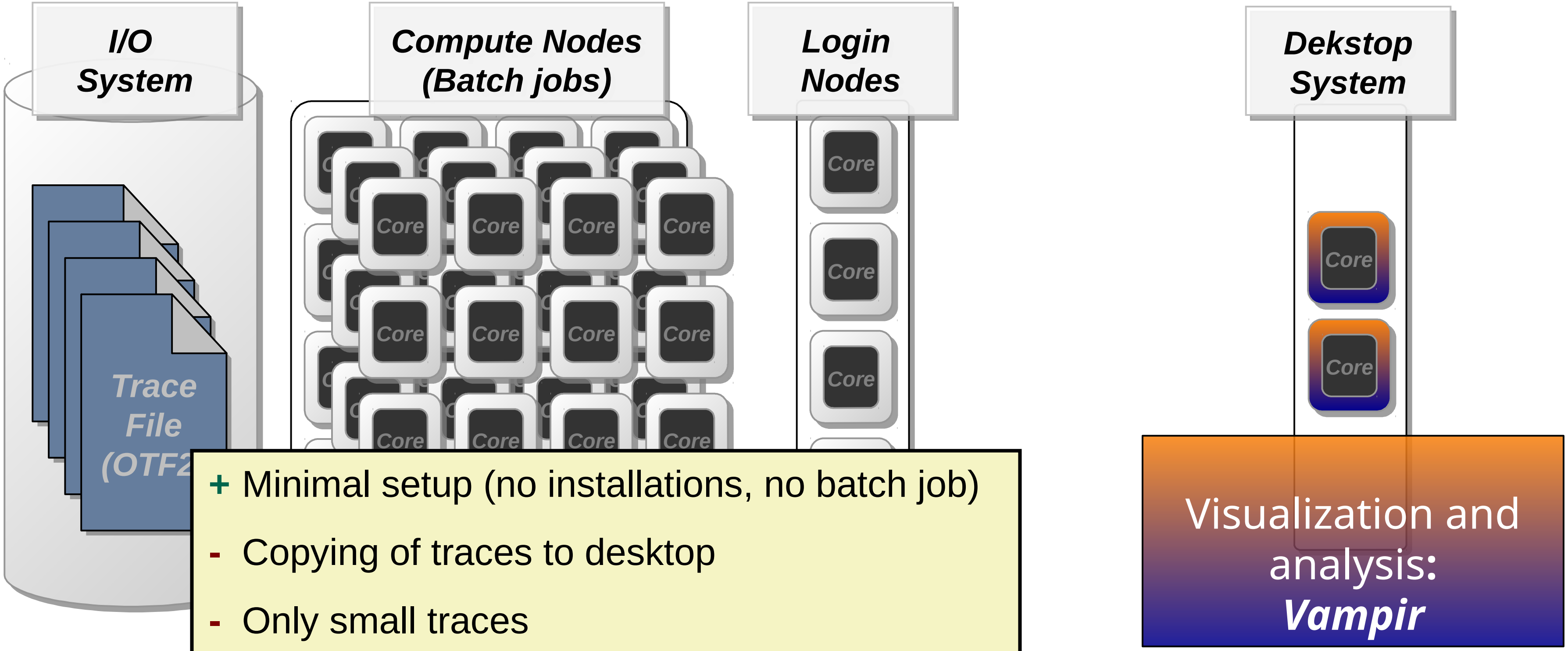
Vampir

Performance data visualization in a complex environment



Simplest Approach

Use your destop system

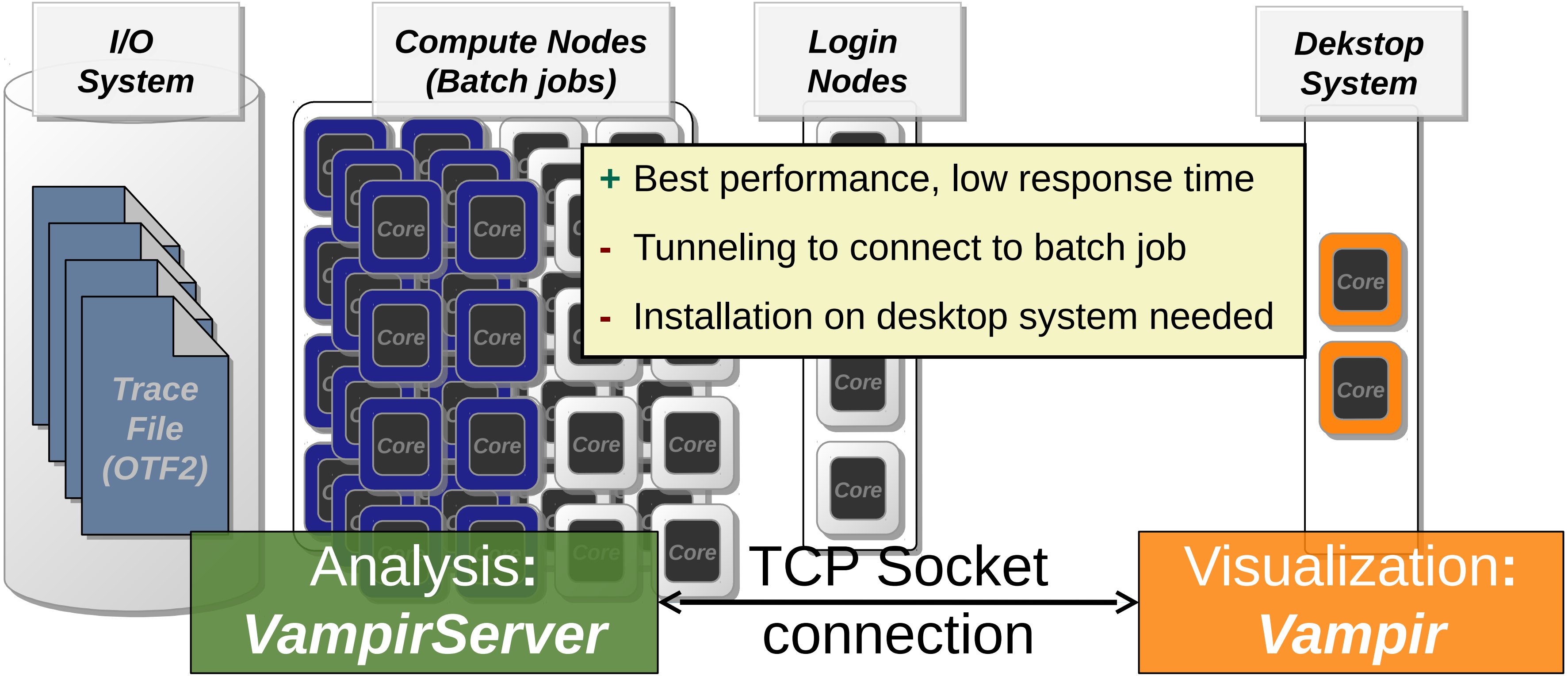


- \+ Minimal setup (no installations, no batch job)
- \- Copying of traces to desktop
- \- Only small traces

(Re)Using the HPC Resources

Run analysis engine on compute nodes, GUI on desktop

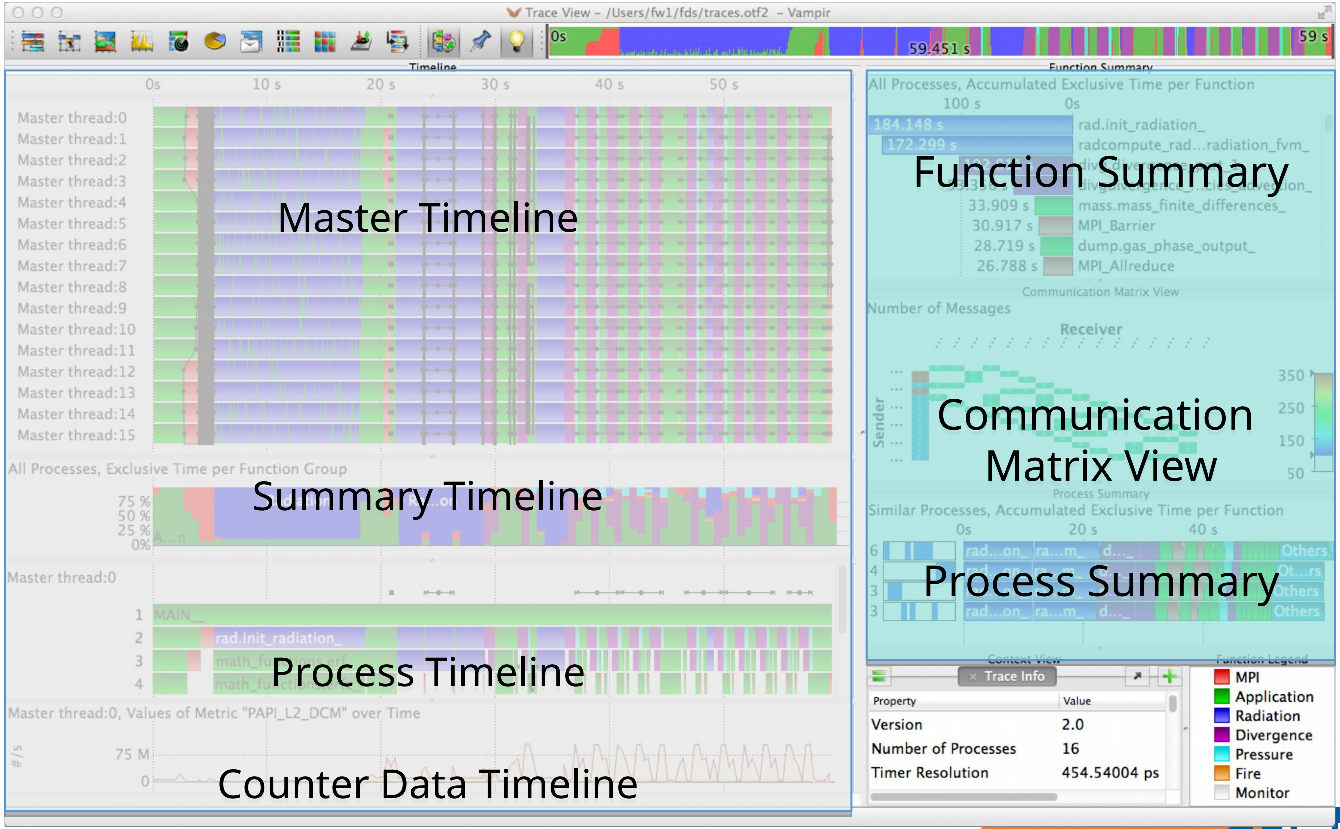
NVIDIA GPU CENTER OF EXCELLENCE



- \+ Best performance, low response time
- \- Tunneling to connect to batch job
- \- Installation on desktop system needed

Vampir GUI

What do the fancy colors mean?



GPU CENTER OF EXCELLENCE

Vampir GUI

Timeline Charts

Master Timeline	➡	*all threads' activities over time per thread*
Summary Timeline	➡	*all threads' activities over time per activity*
Performance Radar	➡	*all threads' perf-metric over time*
Process Timeline	➡	*single thread's activities over time*
Counter Data Timeline	➡	*single threads perf-metric over time*

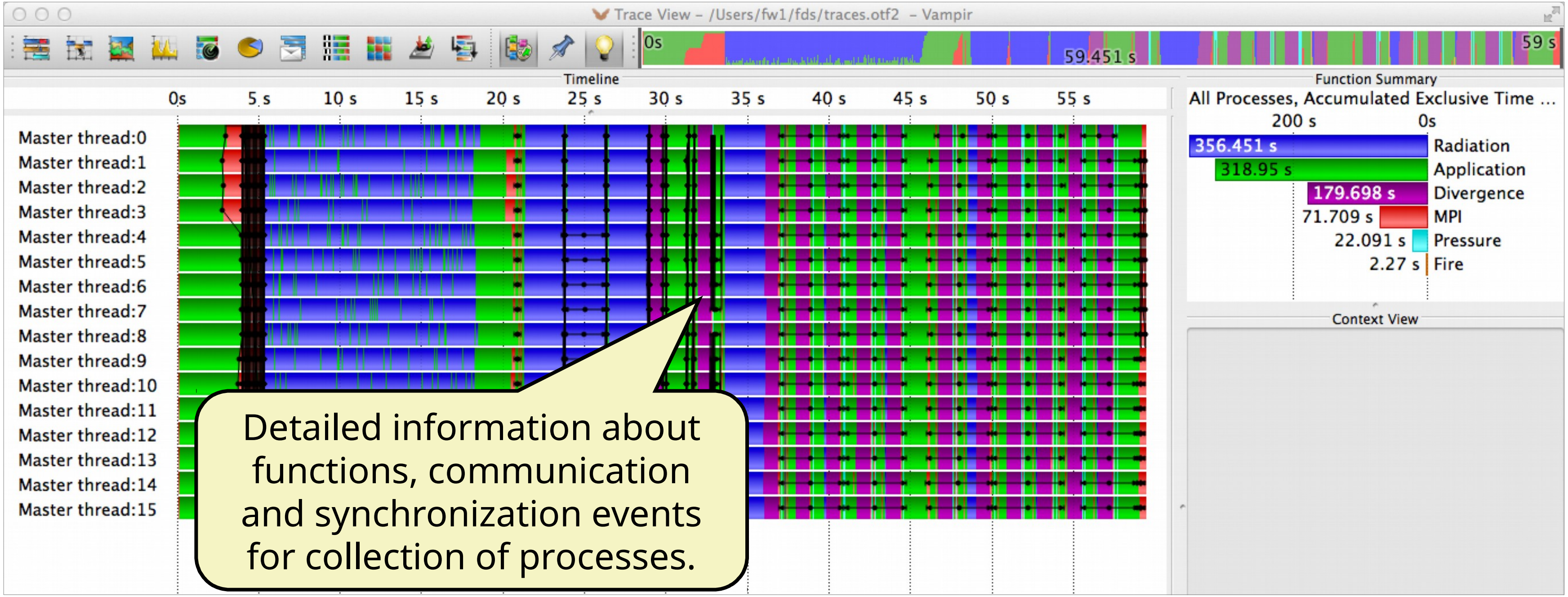
Vampir GUI

Summary/Profile Charts

- Function Summary ➡ *runtime/invocation summaries*
- Message Summary ➡ *data transfer statistics*
- I/O Summary ➡ *I/O statistics*
- Process Summary ➡ *Clustering of similar event streams*
- Communication Matrix View ➡ *Pairwise communincation statistics*

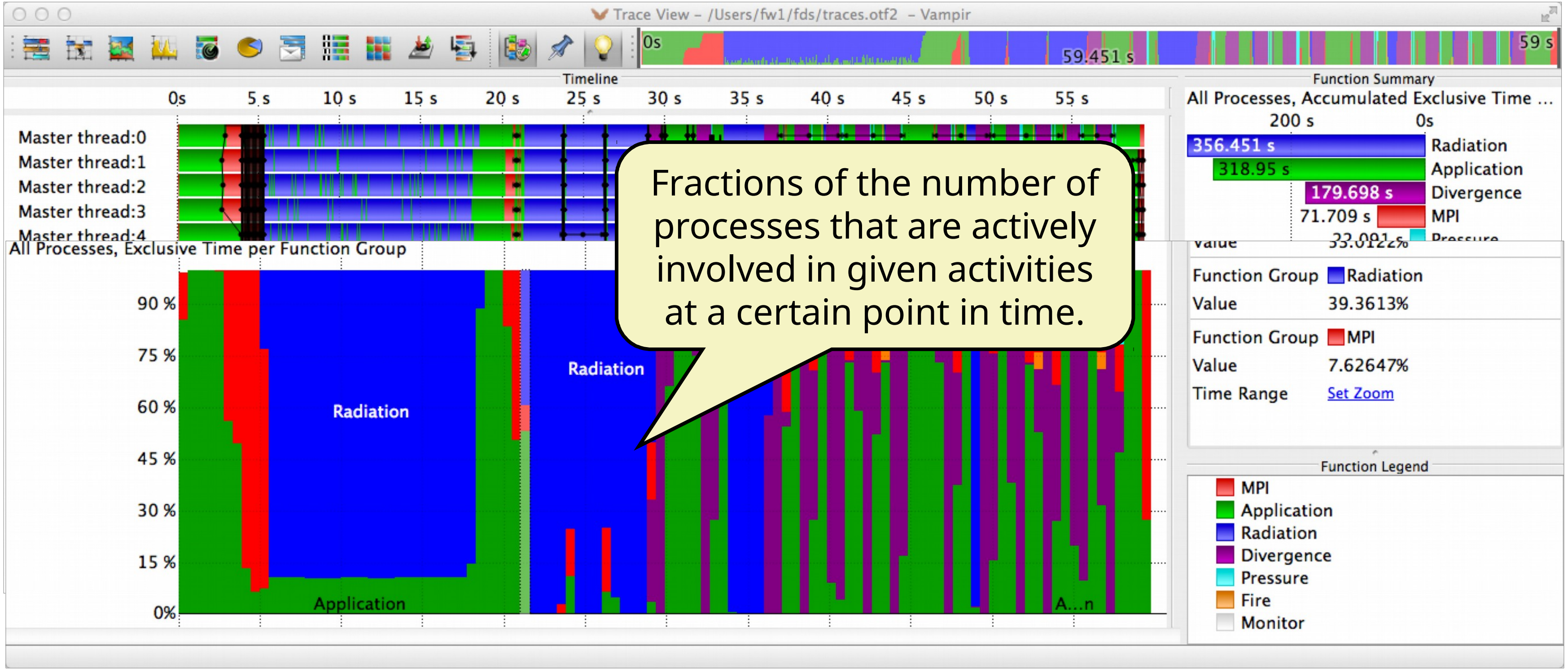
Vampir Performance Charts in Detail

Master Timeline



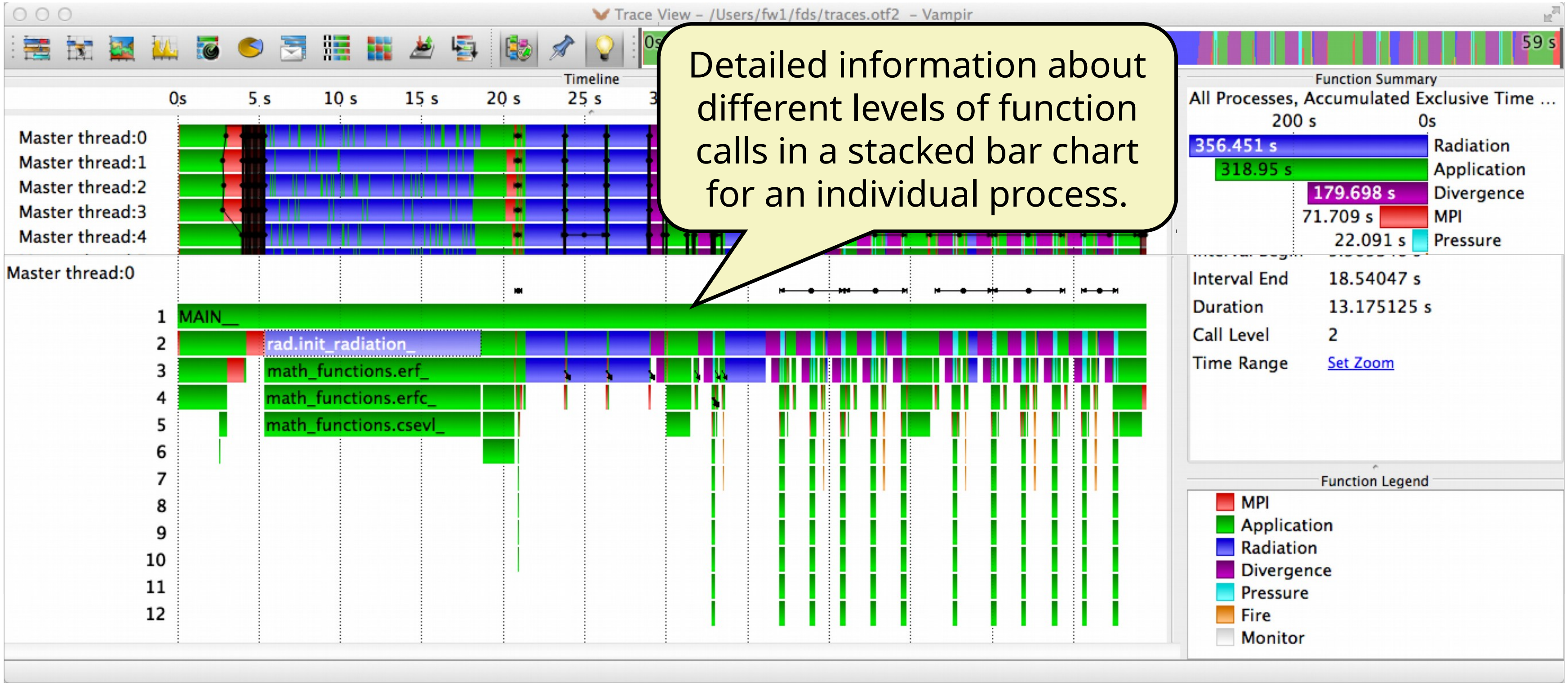
Vampir Performance Charts in Detail

Summary Timeline



Vampir Performance Charts in Detail

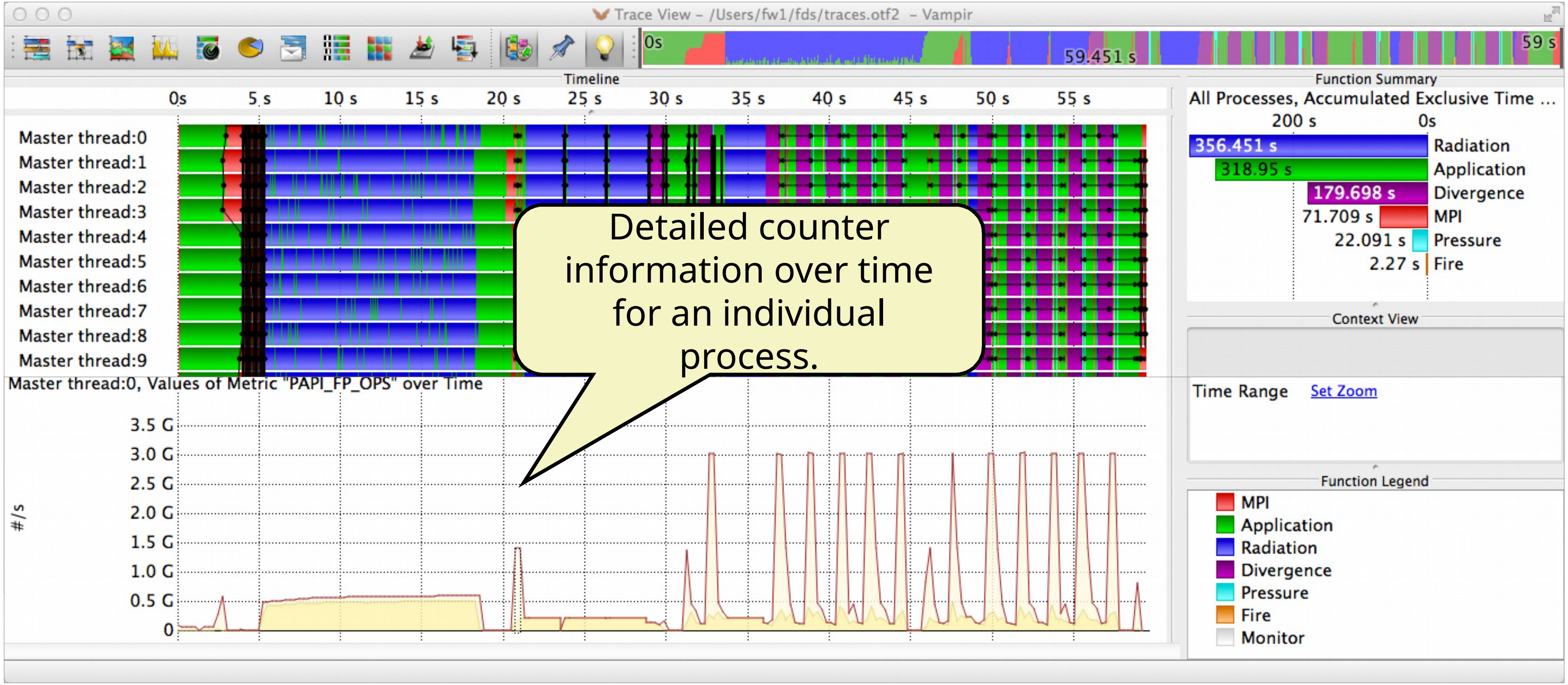
Process Timeline



Vampir Performance Charts in Detail

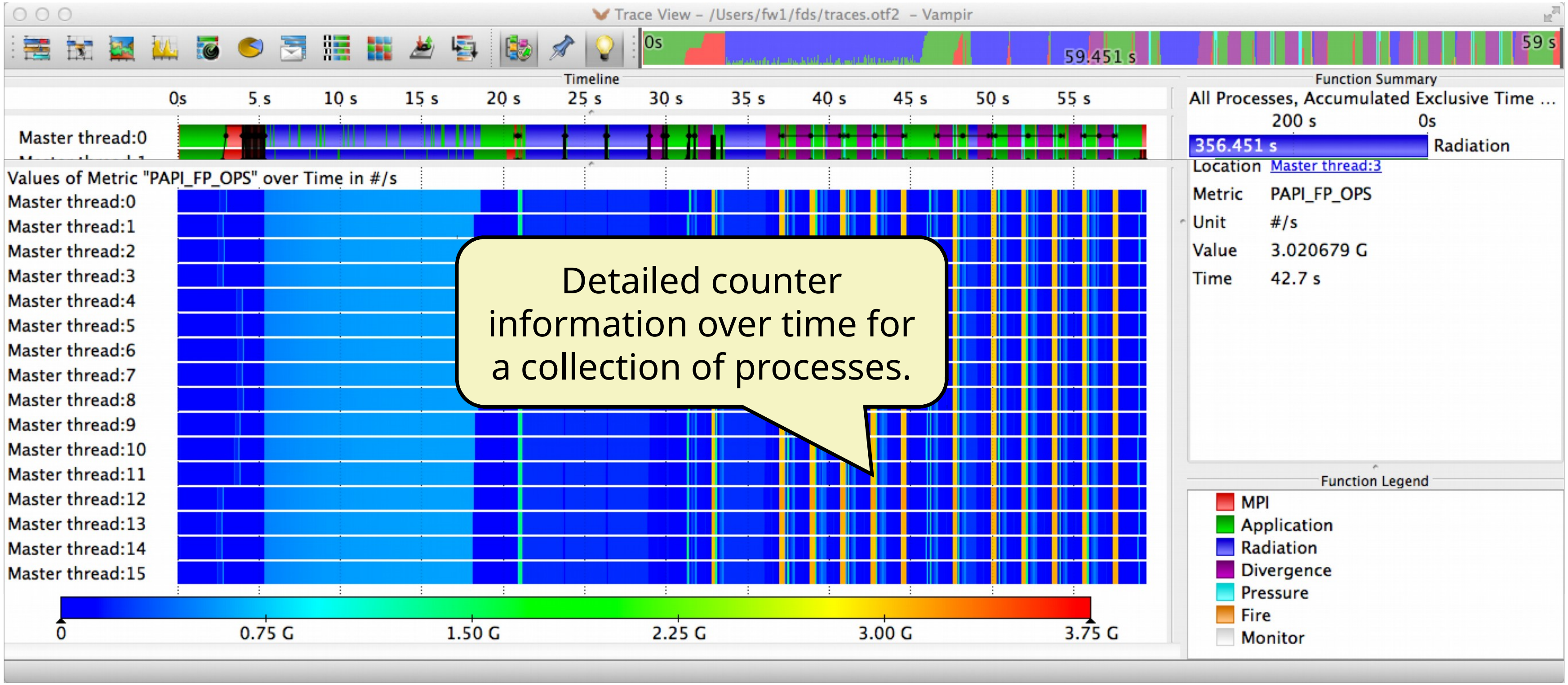
Counter Timeline

GPU CENTER OF EXCELLENCE



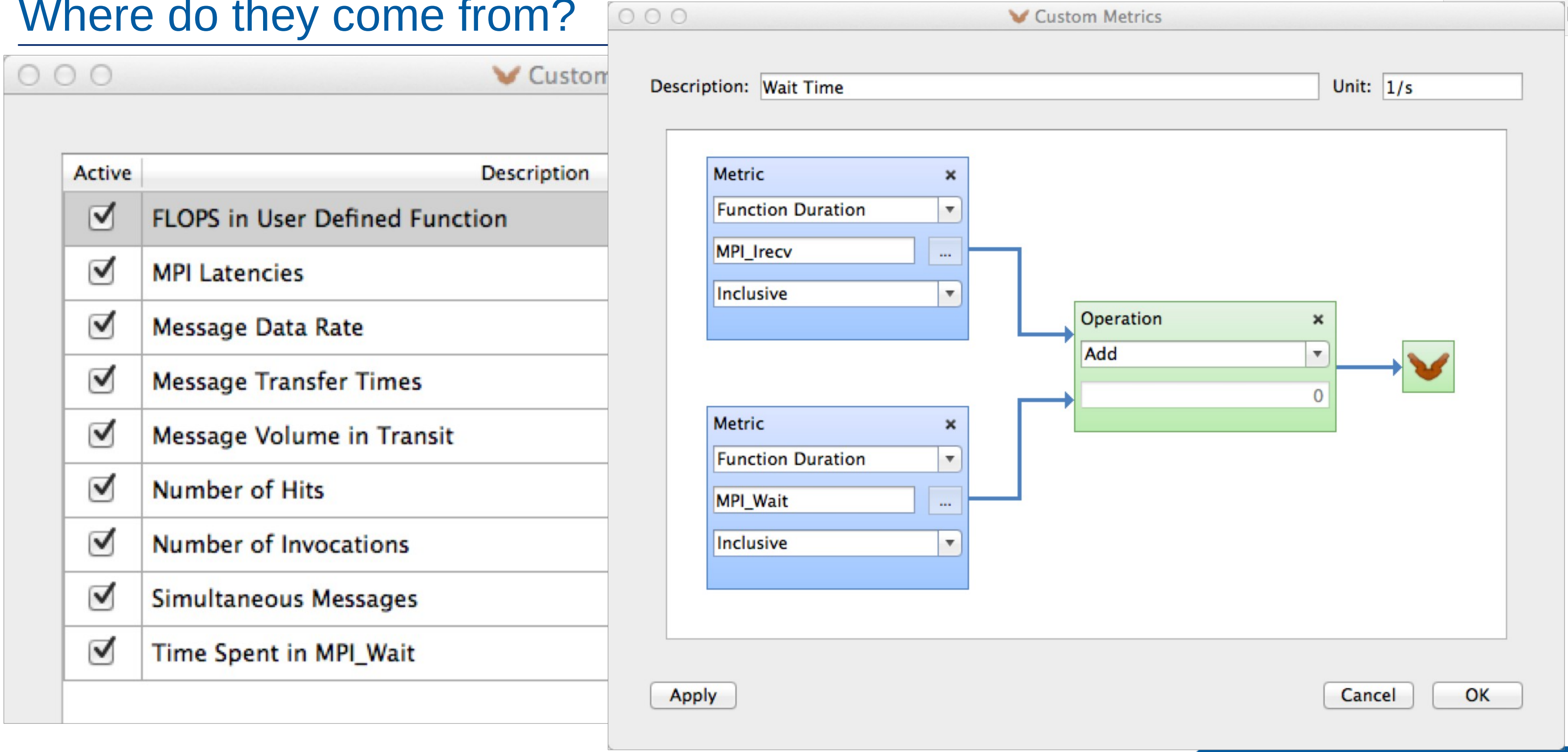
Vampir Performance Charts in Detail

Performance Radar



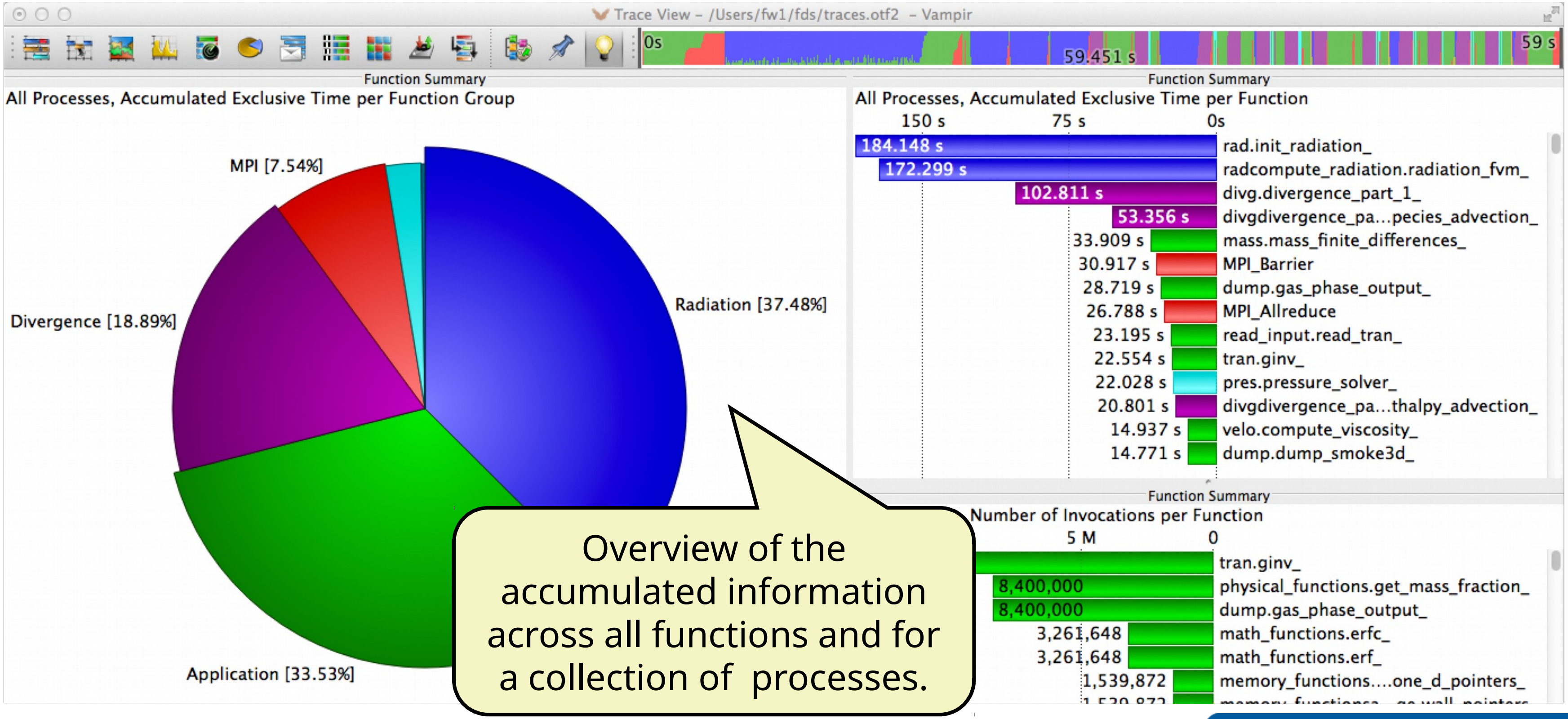
Vampir Performance Metrics

Where do they come from?



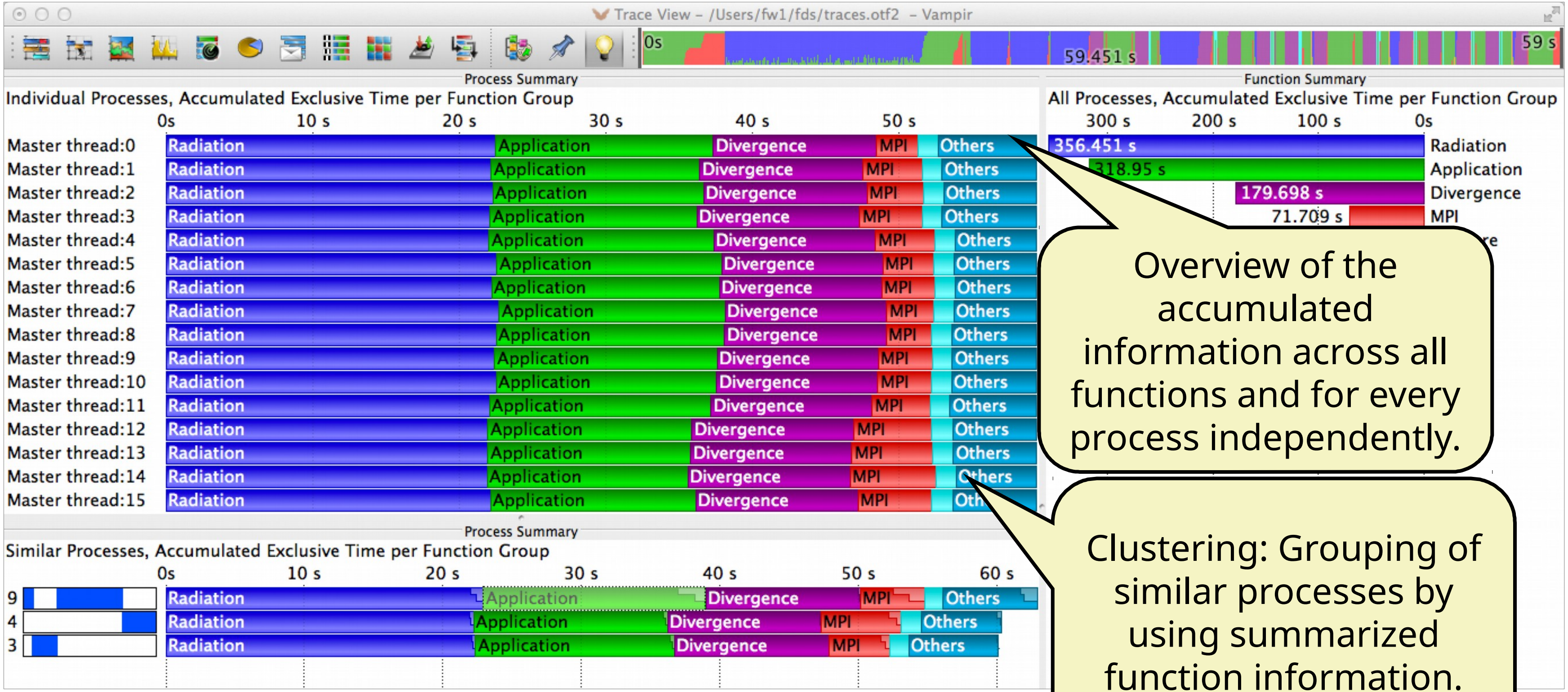
Vampir Performance Charts in Detail

Function Summary



Vampir Performance Charts in Detail

Process Summary

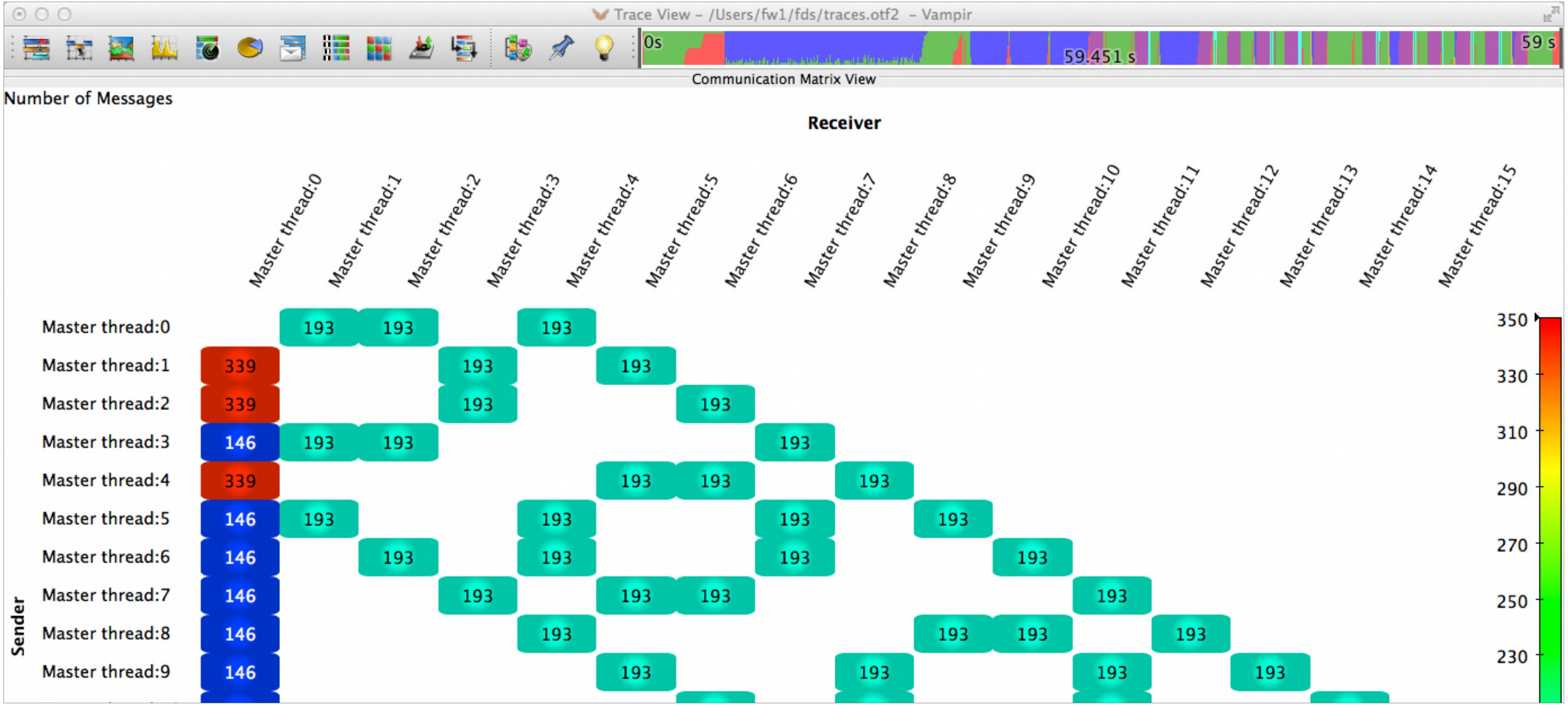


Overview of the accumulated information across all functions and for every process independently.

Clustering: Grouping of similar processes by using summarized function information.

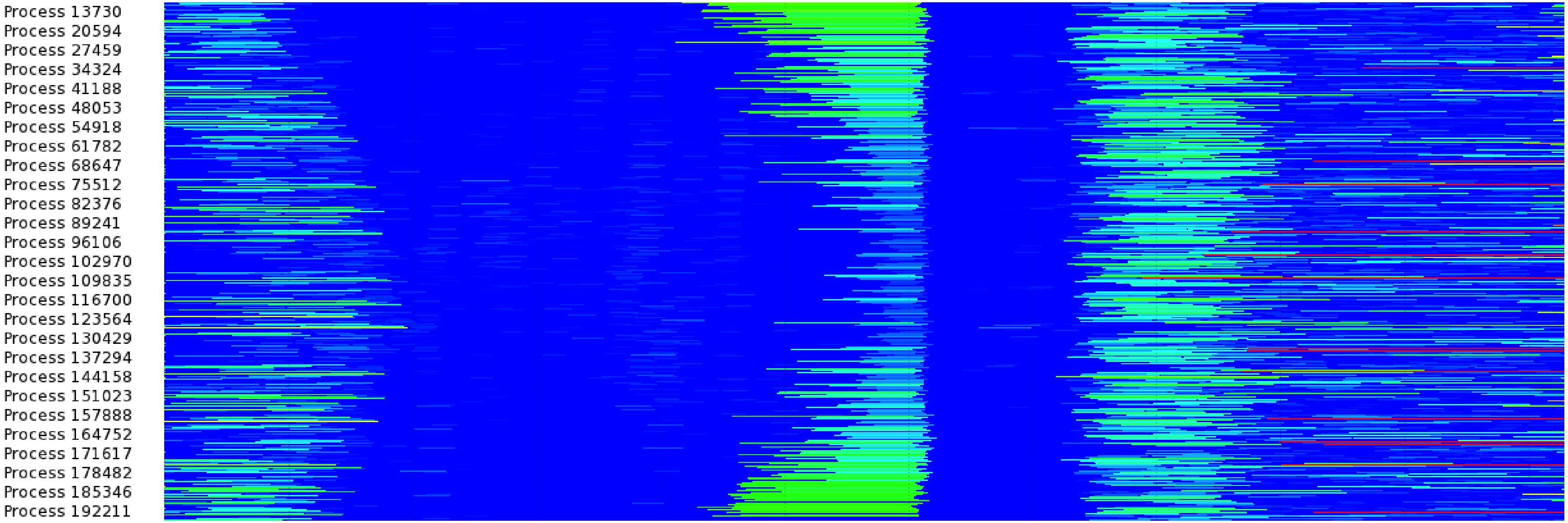
Vampir Performance Charts in Detail

Communication Matrix View

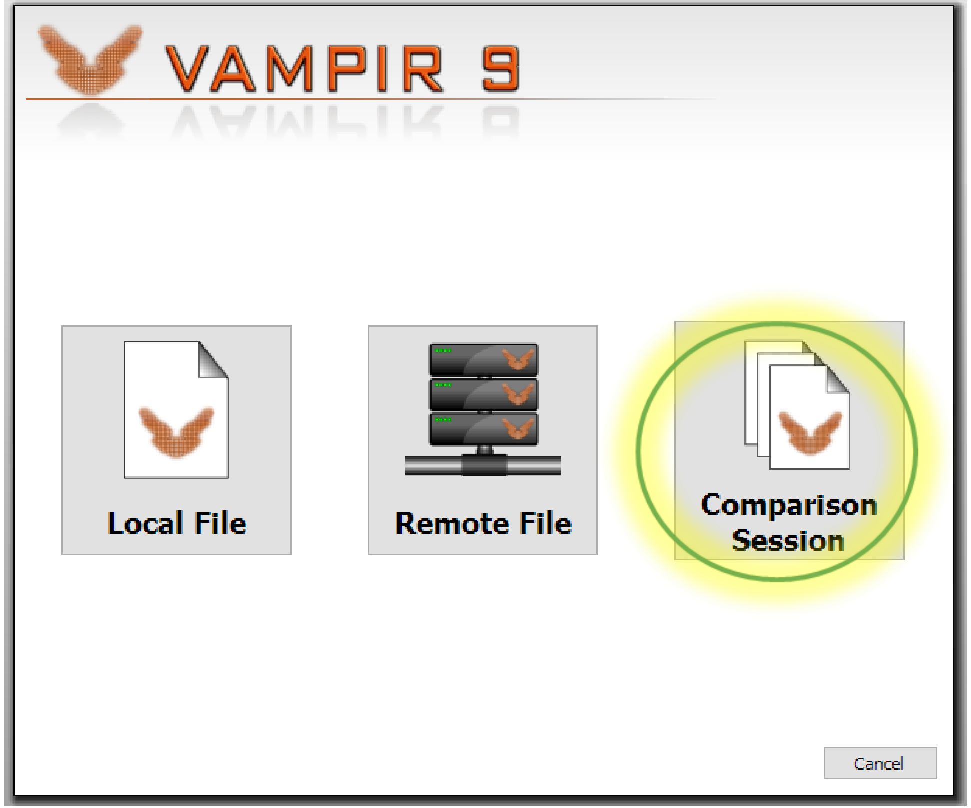


Vampir at Scale

Fit to chart height (feat. 200,000+ event streams)

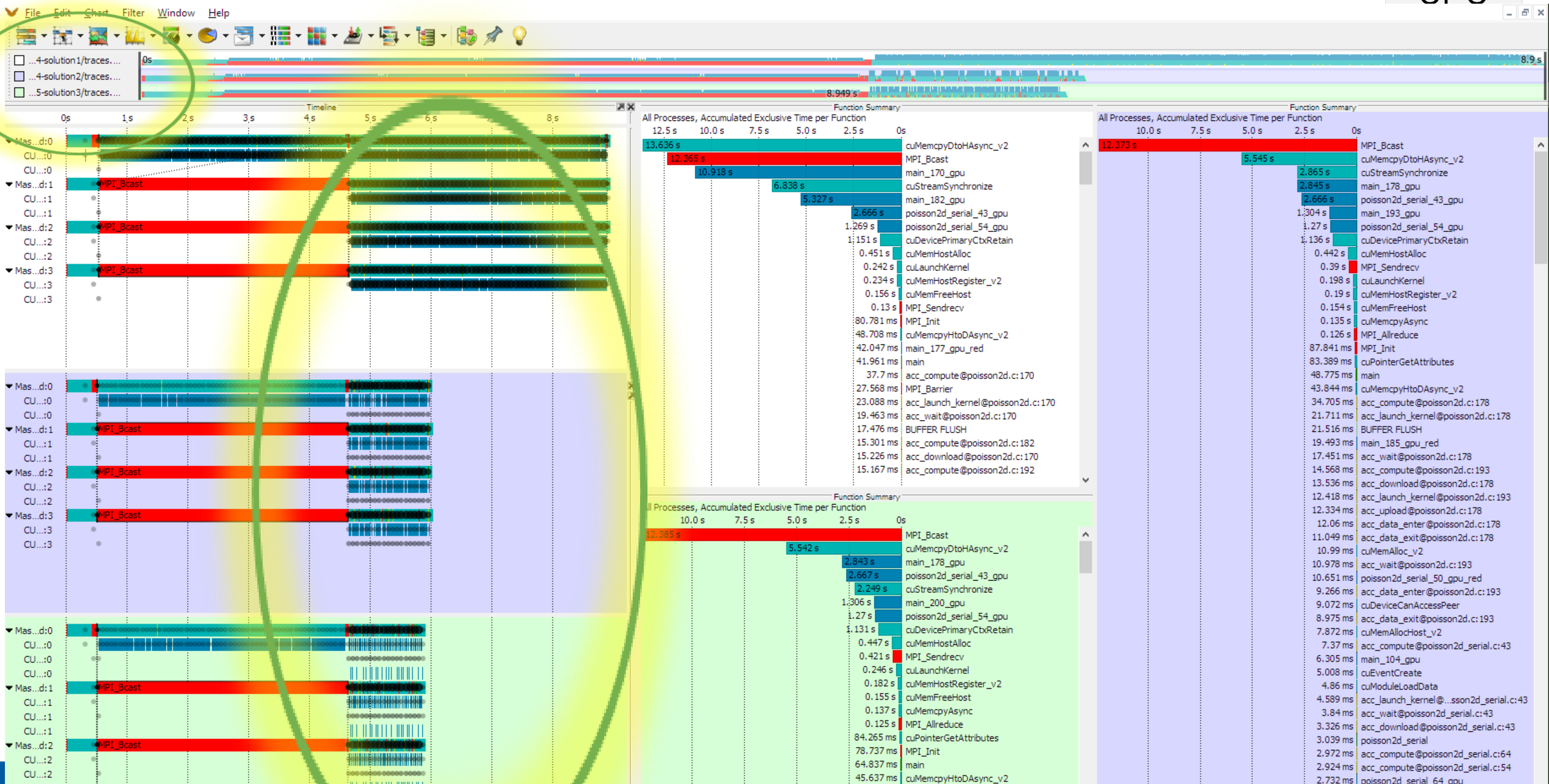


Comparing Traces with Vampir

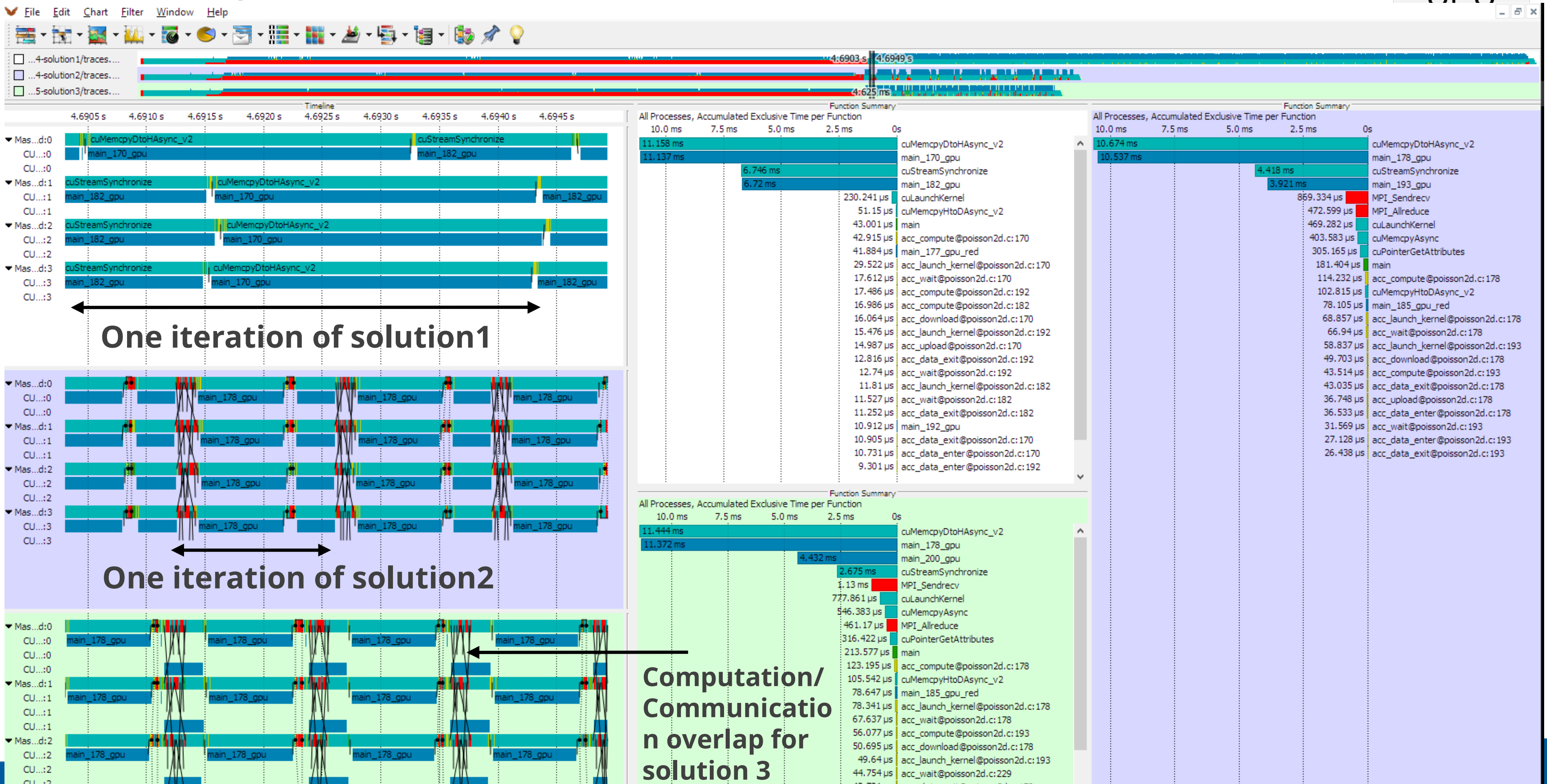


Seeing the differences

GPU



Zooming in


File Edit Chart Filter Window Help
...4-solution1/traces....
...4-solution2/traces....
...5-solution3/traces....
4.6903 s
4.6949 s
4.625 ms
Timeline
4.6905 s 4.6910 s 4.6915 s 4.6920 s 4.6925 s 4.6930 s 4.6935 s 4.6940 s 4.6945 s
One iteration of solution1
One iteration of solution2
Computation/ Communication overlap for solution 3
Function Summary
All Processes, Accumulated Exclusive Time per Function
10.0 ms 7.5 ms 5.0 ms 2.5 ms 0s
11.158 ms cuMemcpyDtoHAsync_v2
11.137 ms main_170_gpu
6.746 ms cuStreamSynchronize
6.72 ms main_182_gpu
230.241 µs cuLaunchKernel
51.15 µs cuMemcpyHtoDAsync_v2
43.001 µs main
42.915 µs acc_compute@poisson2d.c:170
41.884 µs main_177_gpu_red
29.522 µs acc_launch_kernel@poisson2d.c:170
17.612 µs acc_wait@poisson2d.c:170
17.486 µs acc_compute@poisson2d.c:192
16.986 µs acc_compute@poisson2d.c:182
16.064 µs acc_download@poisson2d.c:170
15.476 µs acc_launch_kernel@poisson2d.c:192
14.987 µs acc_upload@poisson2d.c:170
12.816 µs acc_data_exit@poisson2d.c:192
12.74 µs acc_wait@poisson2d.c:192
11.81 µs acc_launch_kernel@poisson2d.c:182
11.527 µs acc_wait@poisson2d.c:182
11.252 µs acc_data_exit@poisson2d.c:182
10.912 µs main_192_gpu
10.905 µs acc_data_exit@poisson2d.c:170
10.731 µs acc_data_enter@poisson2d.c:170
9.301 µs acc_data_enter@poisson2d.c:192
Function Summary
All Processes, Accumulated Exclusive Time per Function
10.0 ms 7.5 ms 5.0 ms 2.5 ms 0s
10.674 ms cuMemcpyDtoHAsync_v2
10.537 ms main_178_gpu
4.418 ms cuStreamSynchronize
3.921 ms main_193_gpu
869.334 µs MPI_Sendrecv
472.599 µs MPI_Allreduce
469.282 µs cuLaunchKernel
403.583 µs cuMemcpyAsync
305.165 µs cuPointerGetAttributes
181.404 µs main
114.232 µs acc_compute@poisson2d.c:178
102.815 µs cuMemcpyHtoDAsync_v2
78.105 µs main_185_gpu_red
68.857 µs acc_launch_kernel@poisson2d.c:178
66.94 µs acc_wait@poisson2d.c:178
58.837 µs acc_launch_kernel@poisson2d.c:193
49.703 µs acc_download@poisson2d.c:178
43.514 µs acc_compute@poisson2d.c:193
43.035 µs acc_data_exit@poisson2d.c:178
36.748 µs acc_upload@poisson2d.c:178
36.533 µs acc_data_enter@poisson2d.c:178
31.569 µs acc_wait@poisson2d.c:193
27.128 µs acc_data_enter@poisson2d.c:193
26.438 µs acc_data_exit@poisson2d.c:193
Function Summary
All Processes, Accumulated Exclusive Time per Function
10.0 ms 7.5 ms 5.0 ms 2.5 ms 0s
11.444 ms cuMemcpyDtoHAsync_v2
11.372 ms main_178_gpu
4.432 ms main_200_gpu
2.675 ms cuStreamSynchronize
1.13 ms MPI_Sendrecv
777.861 µs cuLaunchKernel
546.383 µs cuMemcpyAsync
461.17 µs MPI_Allreduce
316.422 µs cuPointerGetAttributes
213.577 µs main
123.195 µs acc_compute@poisson2d.c:178
105.542 µs cuMemcpyHtoDAsync_v2
78.647 µs main_185_gpu_red
78.341 µs acc_launch_kernel@poisson2d.c:178
67.637 µs acc_wait@poisson2d.c:178
56.077 µs acc_compute@poisson2d.c:193
50.695 µs acc_download@poisson2d.c:178
49.64 µs acc_launch_kernel@poisson2d.c:193
44.754 µs acc_wait@poisson2d.c:229

DEMO:

Visualizing Trace Files with Vampir

Looking into DL Frameworks

Score-P Python Bindings

Tracing/Profiling for all python programs

NVIDIA GPU CENTER OF EXCELLENCE

- Not yet included in main release
- Available on GitHub:
  - https://github.com/score-p/scorep_binding_python
- NSight/nvvp for single node DL frameworks still better (user instrumentation)
- Score-P only choice for MPI-parallel DL frameworks

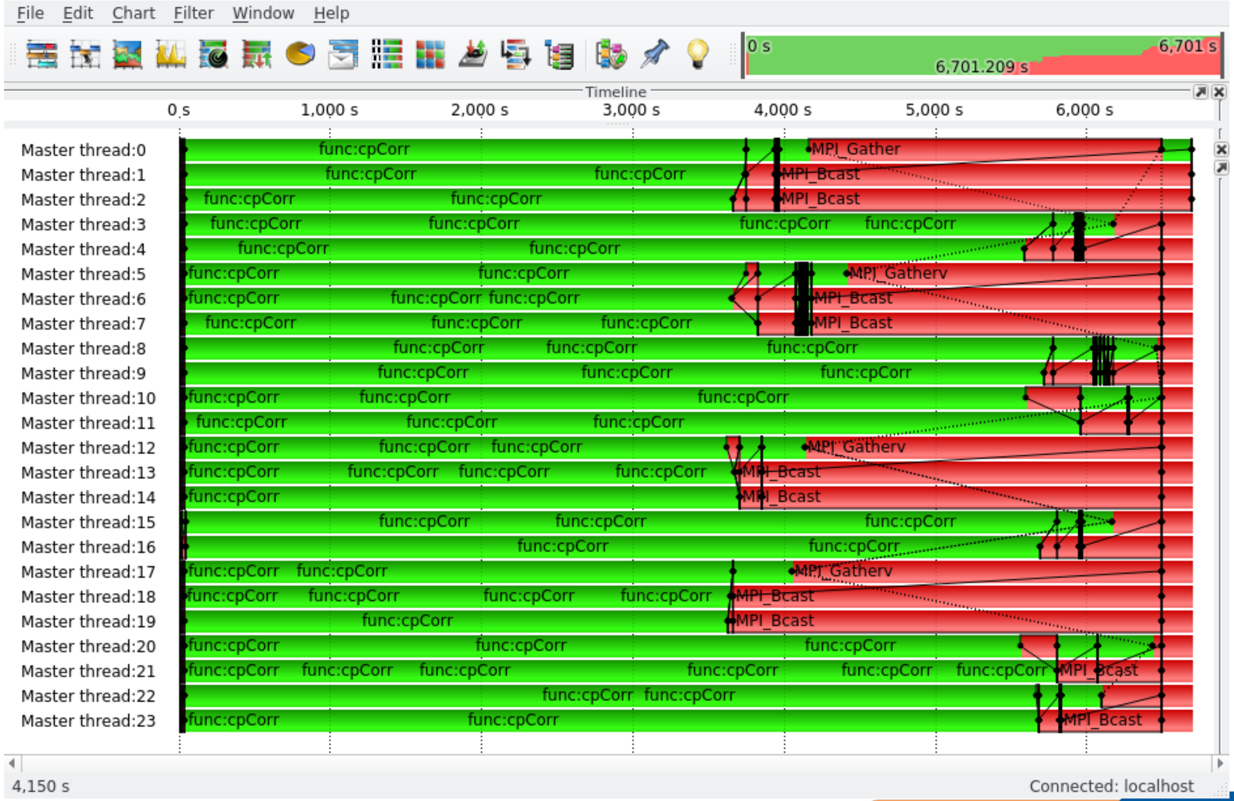
```
$ export SCOREP_ENABLE_PROFILING=true
$ export SCOREP_ENABLE_TRACING=false
$ export SCOREP_EXPERIMENT_DIRECTORY=profile

$ python -m scorep --mpi <script.py>
```

Profiling Example

GPU CENTER OF EXCELLENCE

Vampir with Python Traces

It looks all the same



Vampir is available at http://www.vampir.eu

Vampir at IU: https://kb.iu.edu/d/awbv

Get support via vampirsupport@zih.tu-dresden.de

Score-P: http://www.vi-hps.org/projects/score-p