VAMPIR & VAMPIRTRACE INTRODUCTION AND OVERVIEW

9th VI-HPS Tuning Workshop at UVSQ
April, 2012

Bert Wesarg and Jens Doleschal

Slides by: Andreas Knüpfer, Jens Doleschal,

ZIH, Technische Universität Dresden

Part I: Welcome to the Vampir Tool Suite

- Introduction
- Event Trace Visualization
- Vampir & VampirServer
- The Vampir Displays
    - Timeline
    - Process Timeline with Performance Counters
    - Summary Display
    - Message Statistics
- VampirTrace
    - Instrumentation & Run-Time Measurement

Part II: Hands On

Why bother with performance analysis?

- Well, why are you here after all?
- Efficient usage of expensive and limited resources
- Scalability to achieve next bigger simulation

Profiling and Tracing

- Have an optimization phase
  - just like testing and debugging phase
- Use tools!
- Avoid *do-it-yourself-with-printf* solutions, really!

Trace Visualization

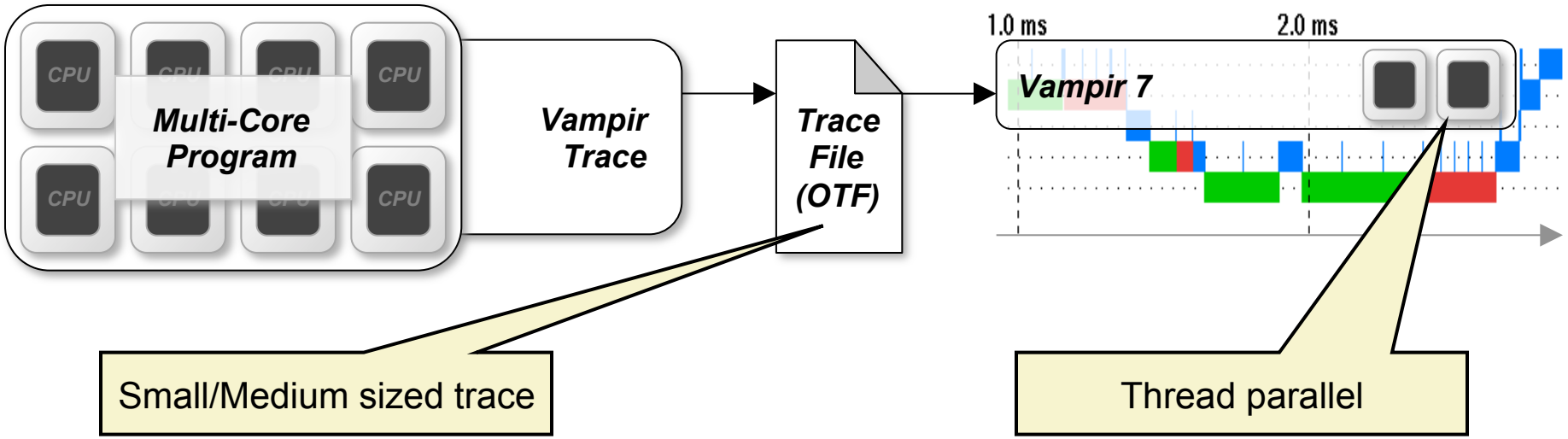
- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior graphically
- Provide statistics and performance metrics
  - Global timeline for parallel processes/threads
  - Process timeline plus performance counters
  - Statistics summary display
  - Message statistics
  - more
- Interactive browsing, zooming, selecting
  - Adapt statistics to zoom level (time interval)
  - Also for very large and highly parallel traces

BUT, it does neither solve your problems automatically nor point you directly at them. It does, however, give you FULL insight into the execution of your application.

Vampir – Visualization Modes (1)

- Directly on front end or local machine

```
% vampir
```

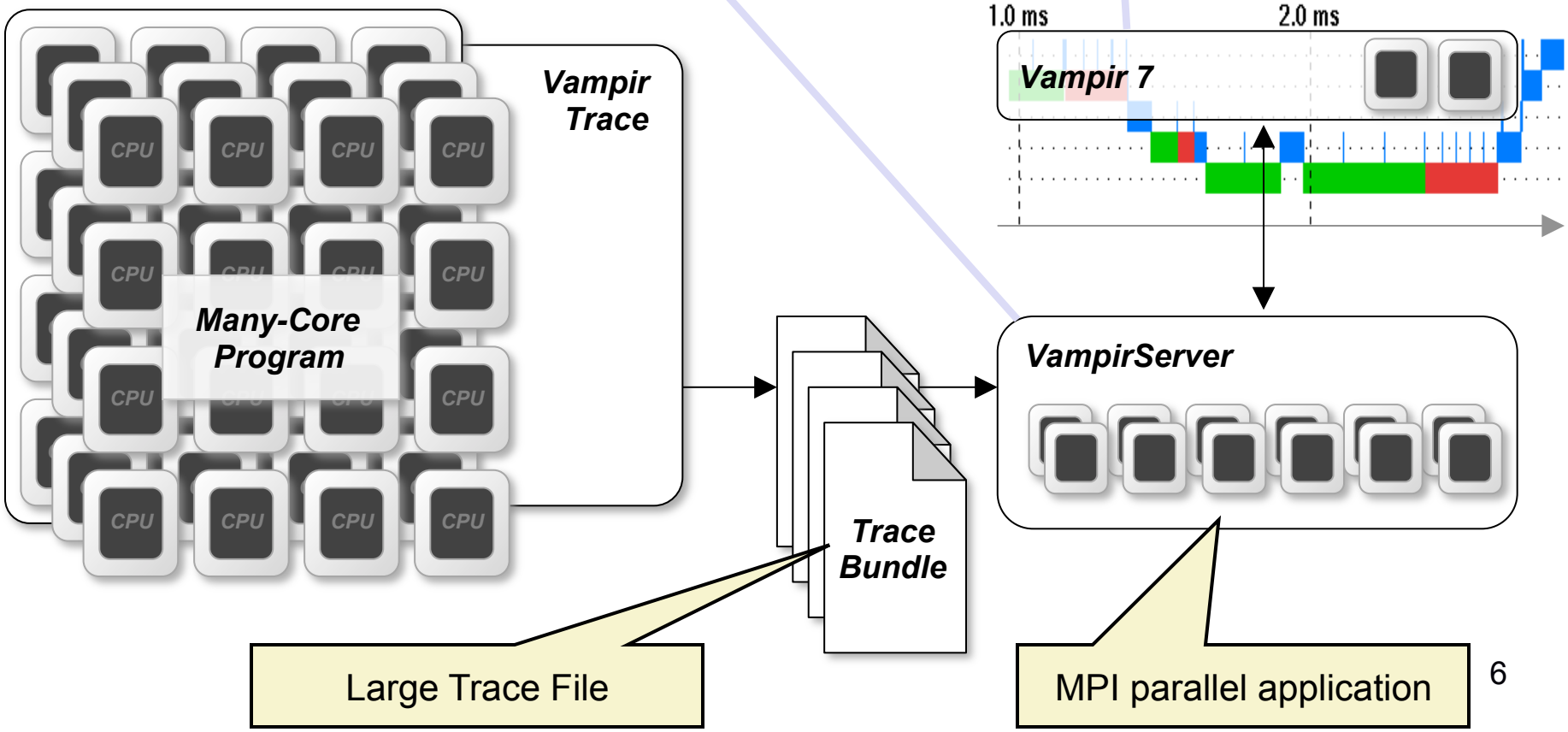


Vampir – Visualization Modes (2)

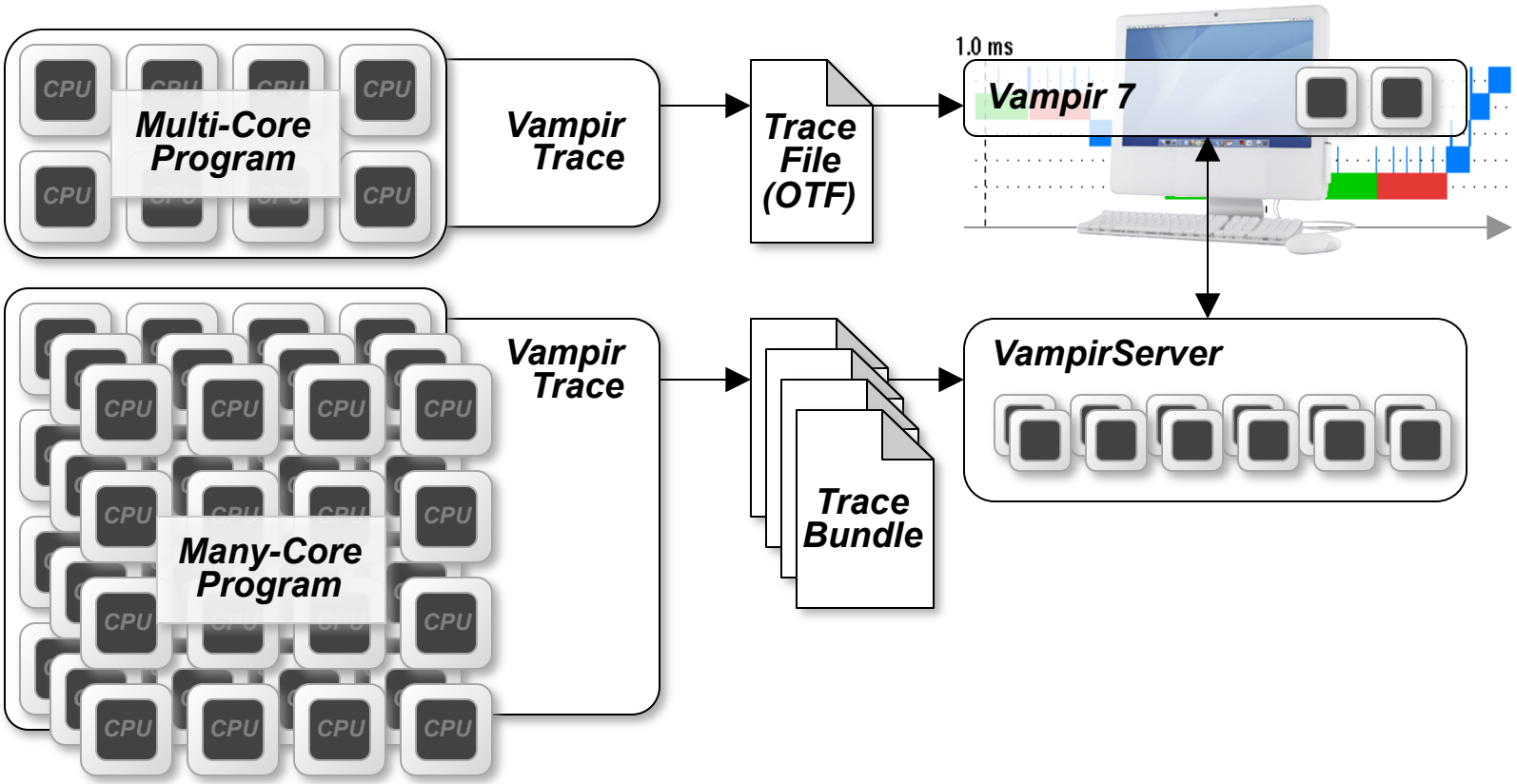
- On local machine with remote VampirServer

```
% vampirserver \
  start -n 12 -p 30088
```

```
% vampir
```



Vampir Toolset Architecture - Overview



Usage order of the Vampir Performance Analysis Toolset

1. Instrument your application with VampirTrace
2. Run your application with an appropriate test set

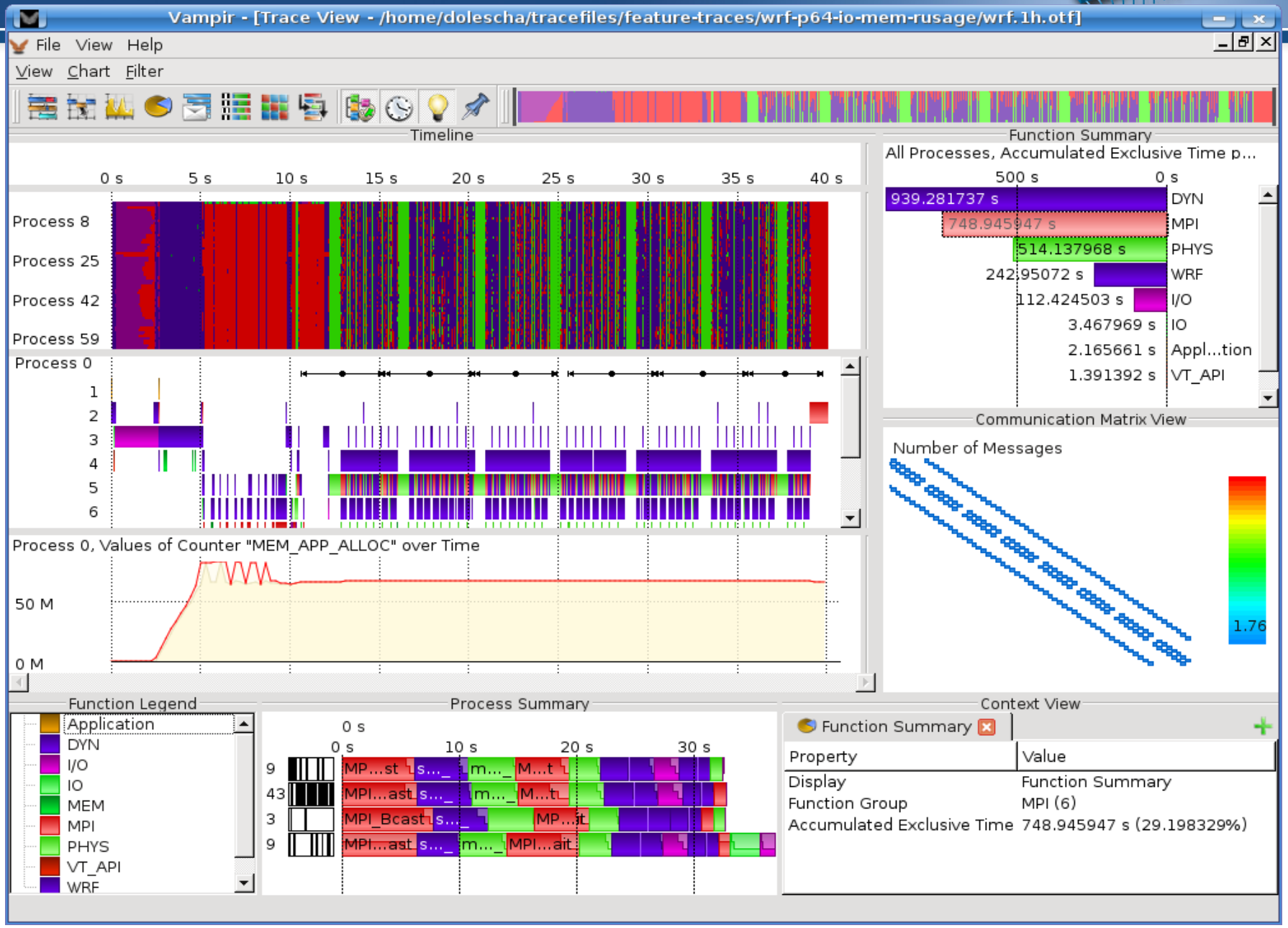
3. Analyze your trace file with Vampir
   - Small trace files can be analyzed on your local workstation
     1. Start your local Vampir
     2. Load trace file from your local disk
   - Large trace files should be stored on the cluster file system
     1. Start VampirServer on your analysis cluster
     2. Start your local Vampir
     3. Connect local Vampir with the VampirServer on the analysis cluster
     4. Load trace file from the cluster file system

Vampir Displays

The main displays of Vampir:

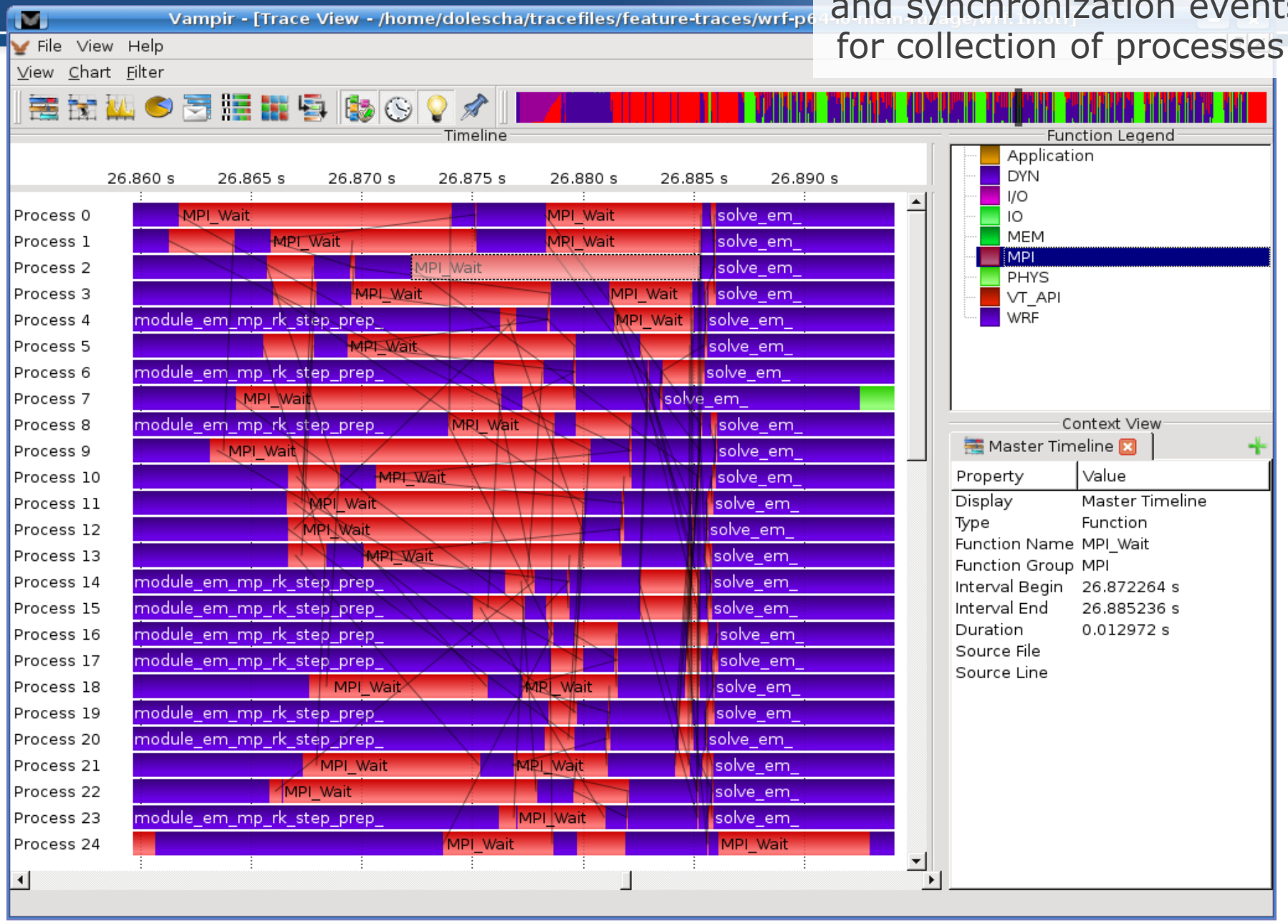
- Master Timeline (Global Timeline)
- Process and Counter Timeline
- Function Summary
- Message Summary
- Process Summary
- Communication Matrix
- Call Tree

Vampir 7: Displays for a WRF Trace with 64 Processes



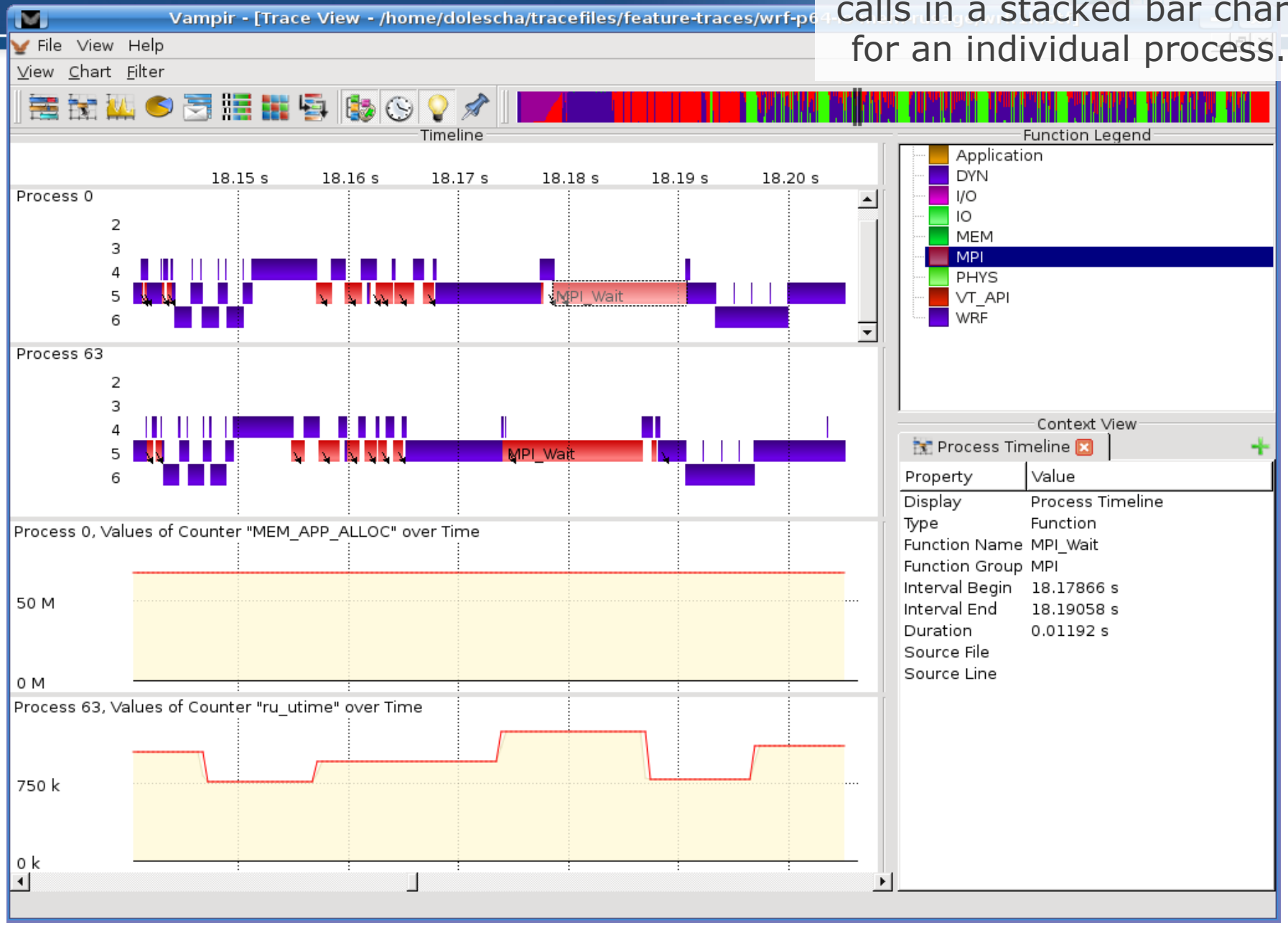
Master Timeline (Global Timeline)

Detailed information about functions, communication and synchronization events for collection of processes.



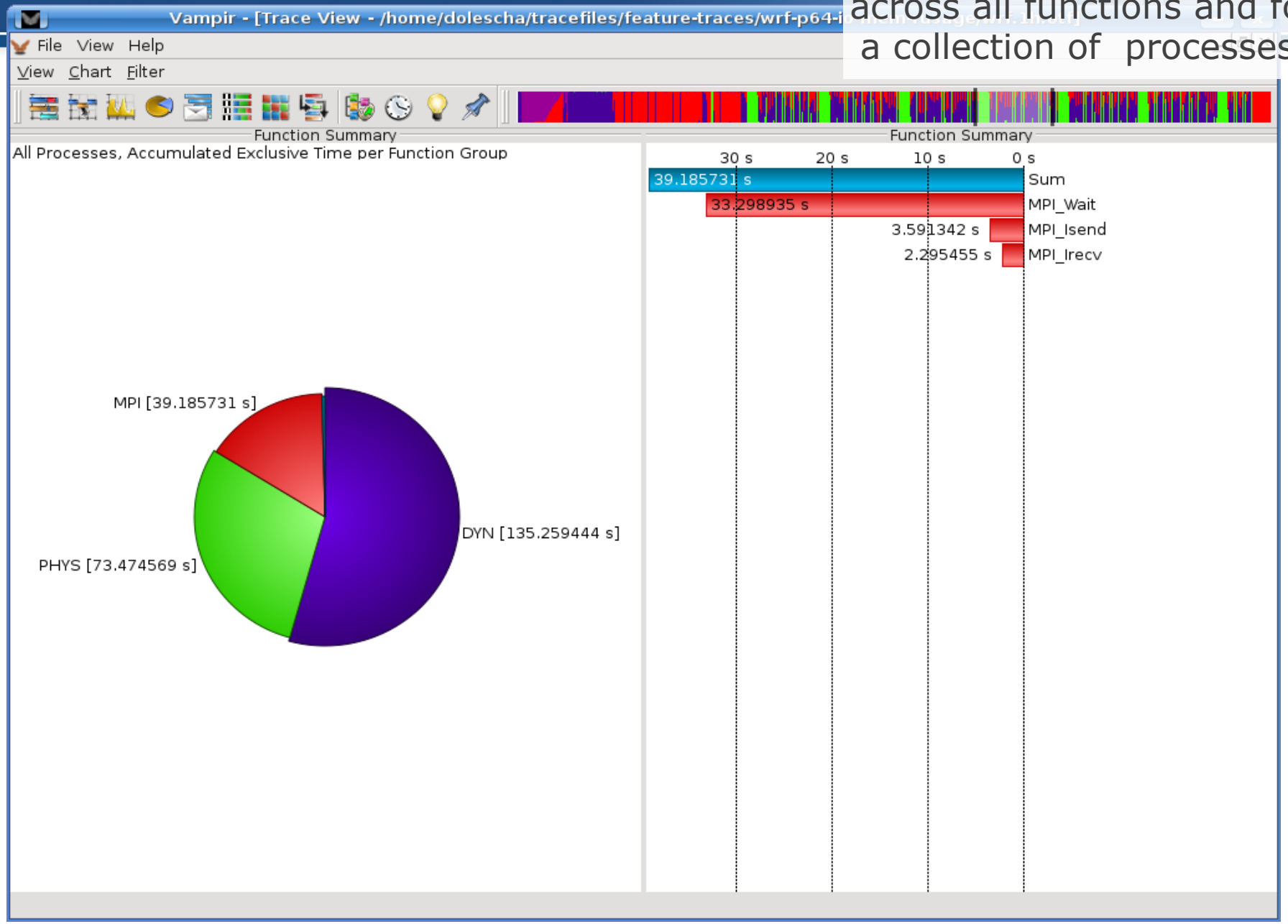
Process and Counter Timeline

Detailed information about different levels of function calls in a stacked bar chart for an individual process.



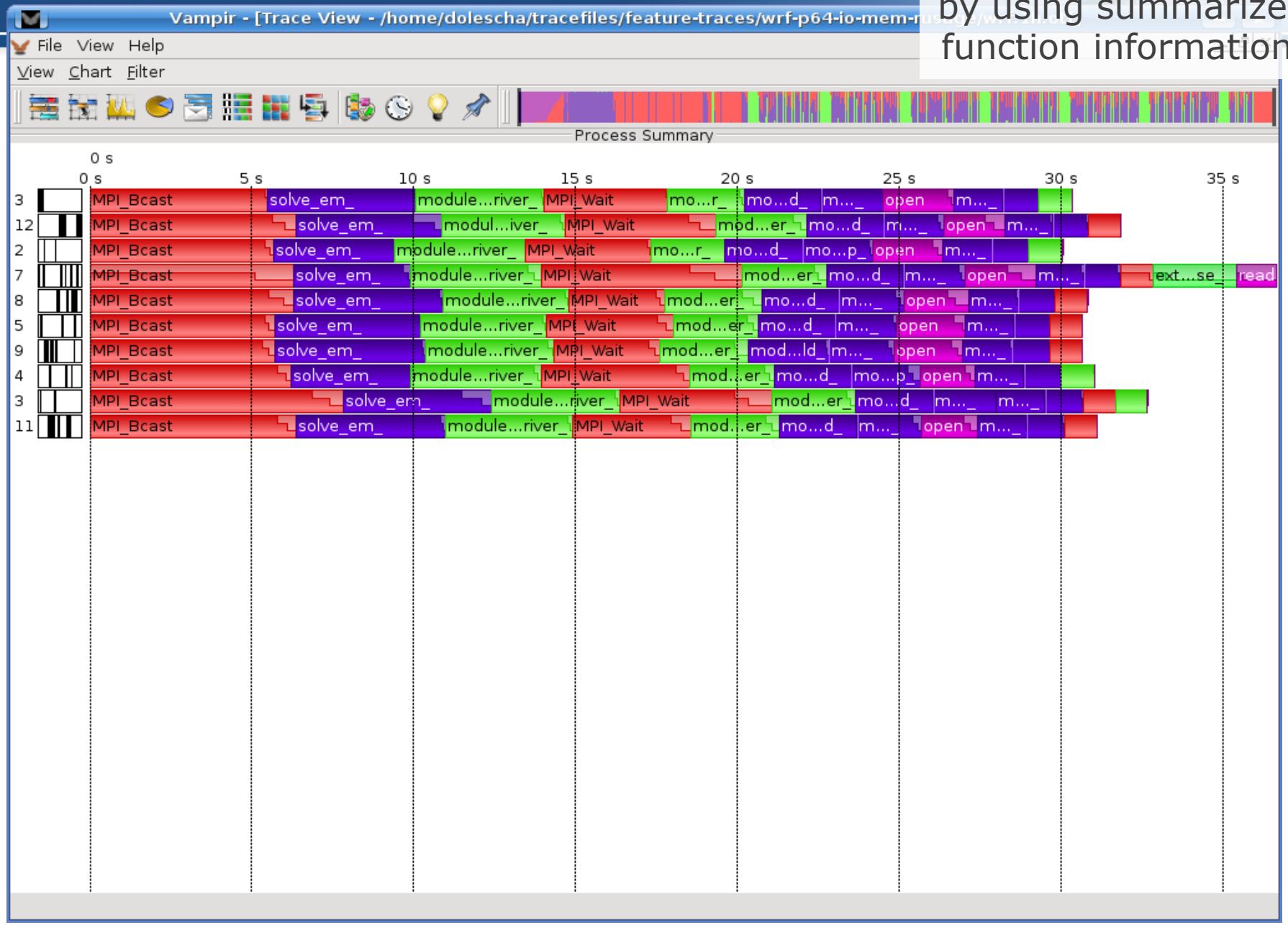
Function Summary

Overview of the accumulated information across all functions and for a collection of processes.



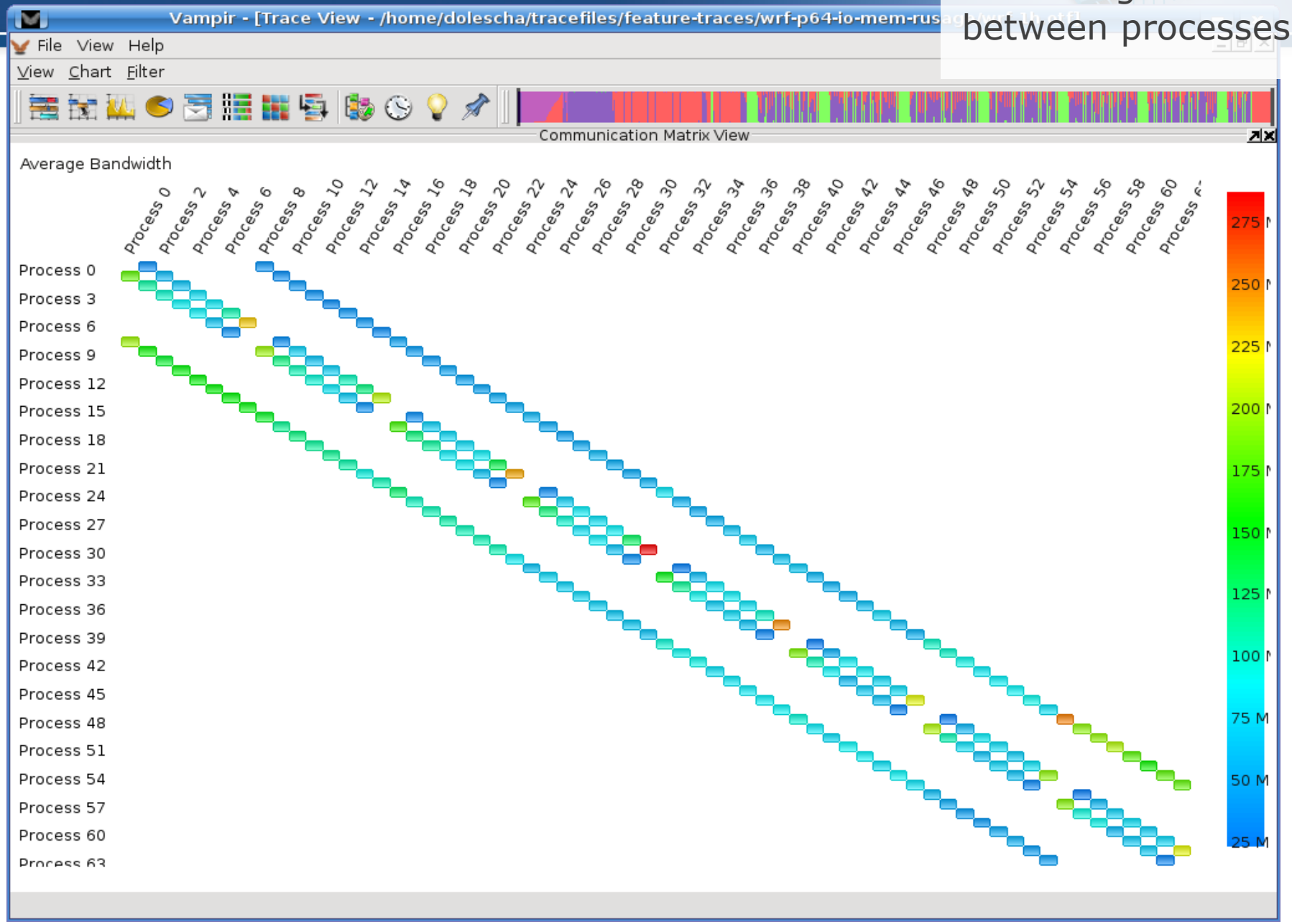
Process Summary

Find groups of similar processes and threads by using summarized function information.



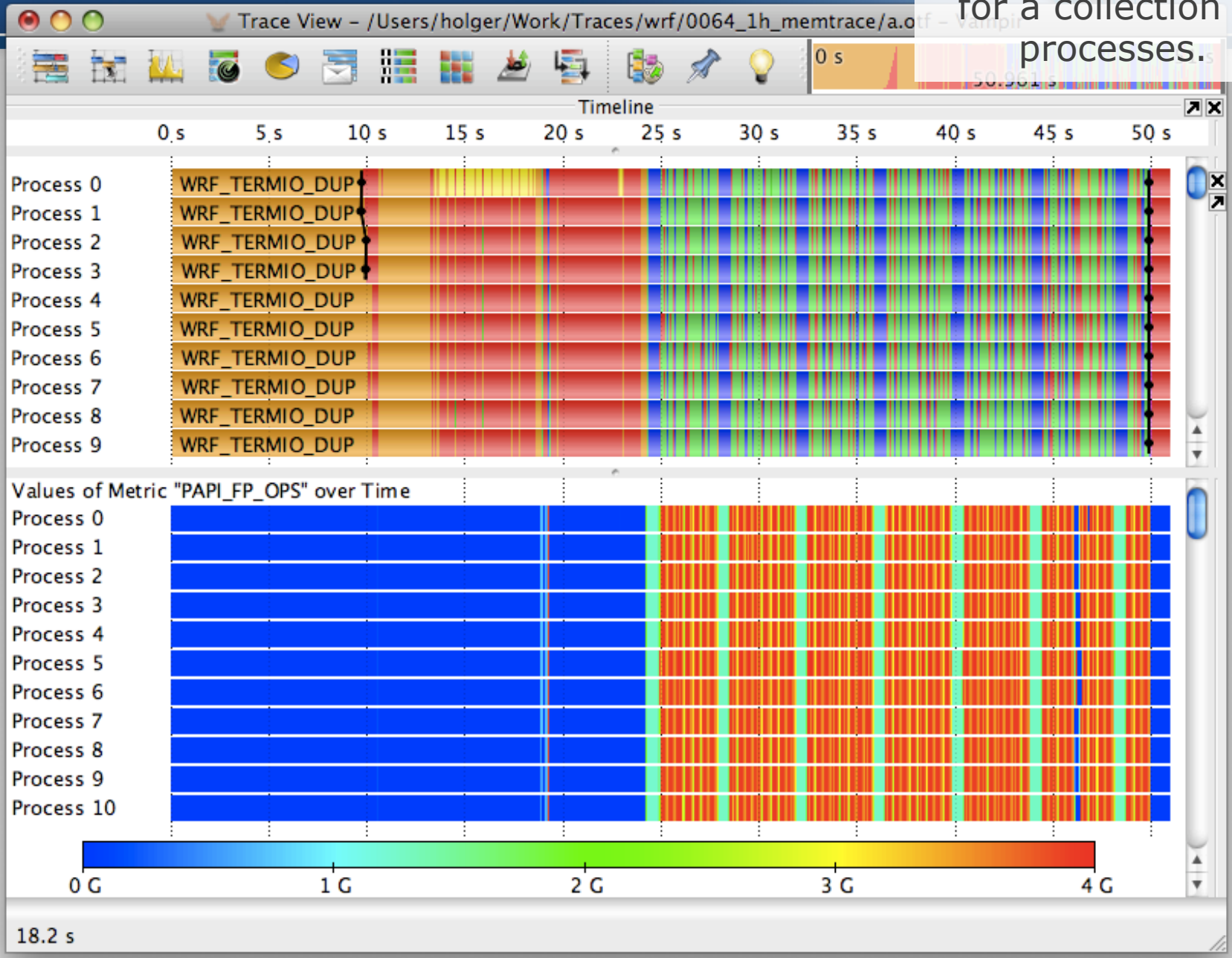
Communication Matrix

Information about messages sent between processes.



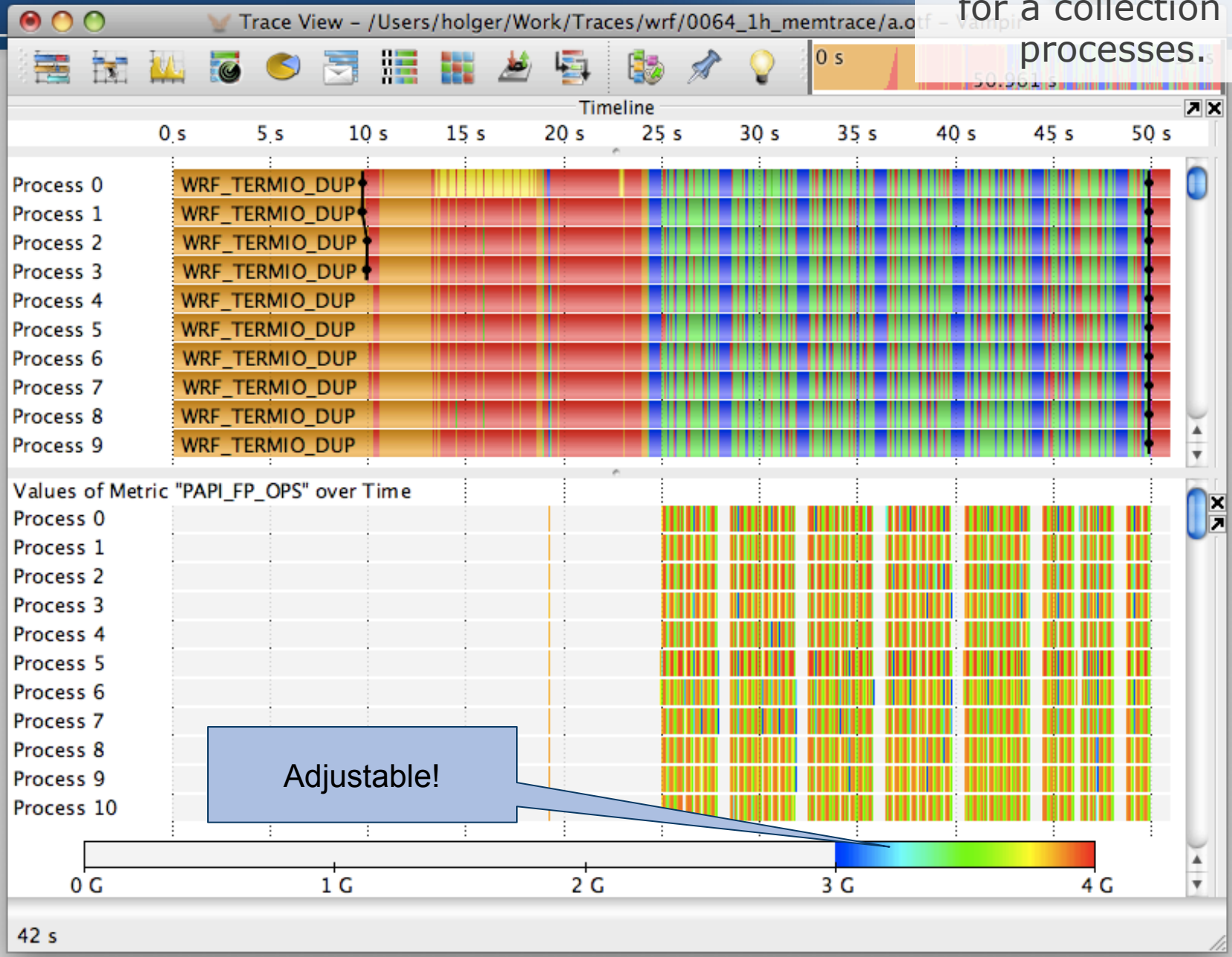
Performance Radar

Detailed counter information over time for a collection of processes.



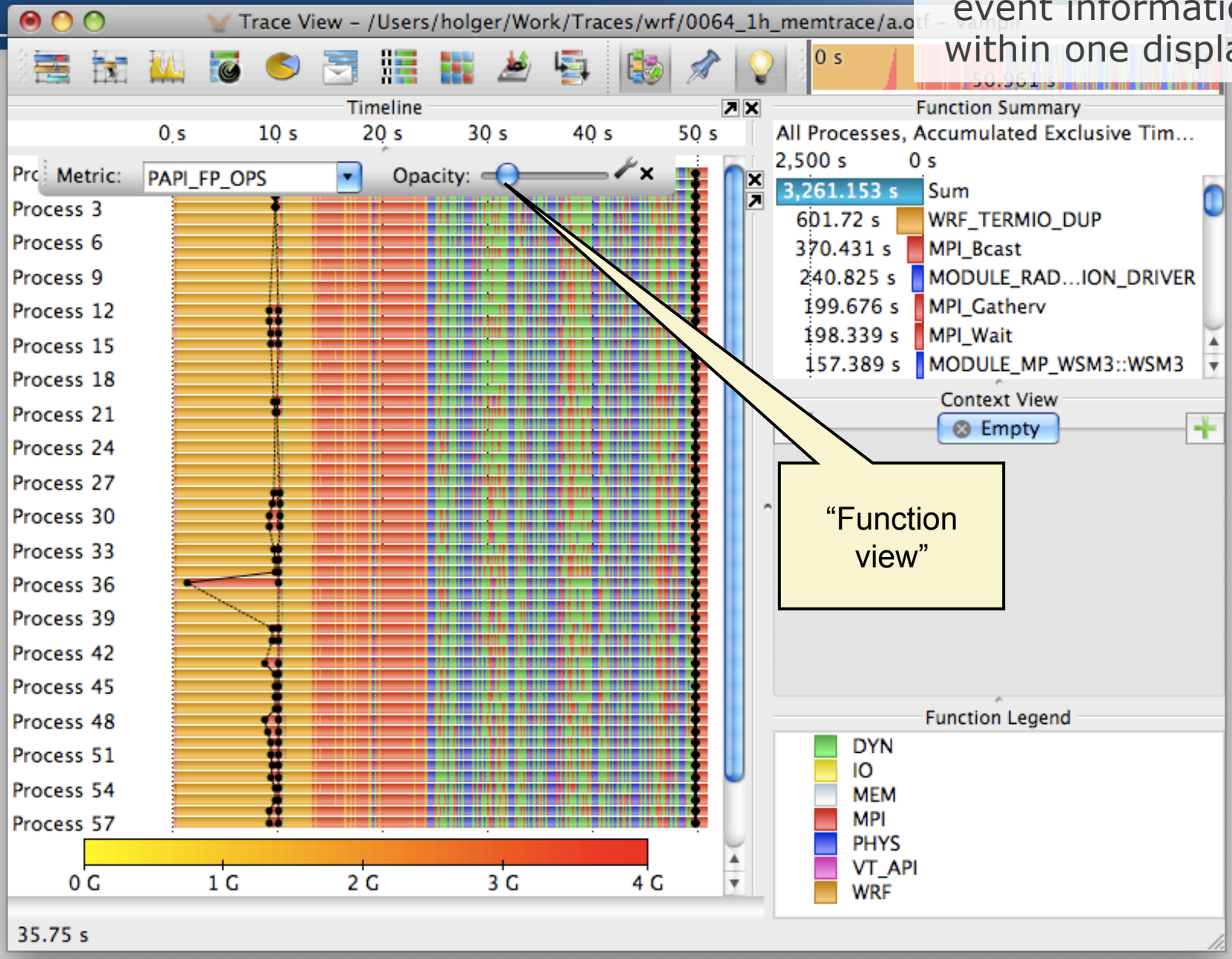
Performance Radar

Detailed counter information over time for a collection of processes.



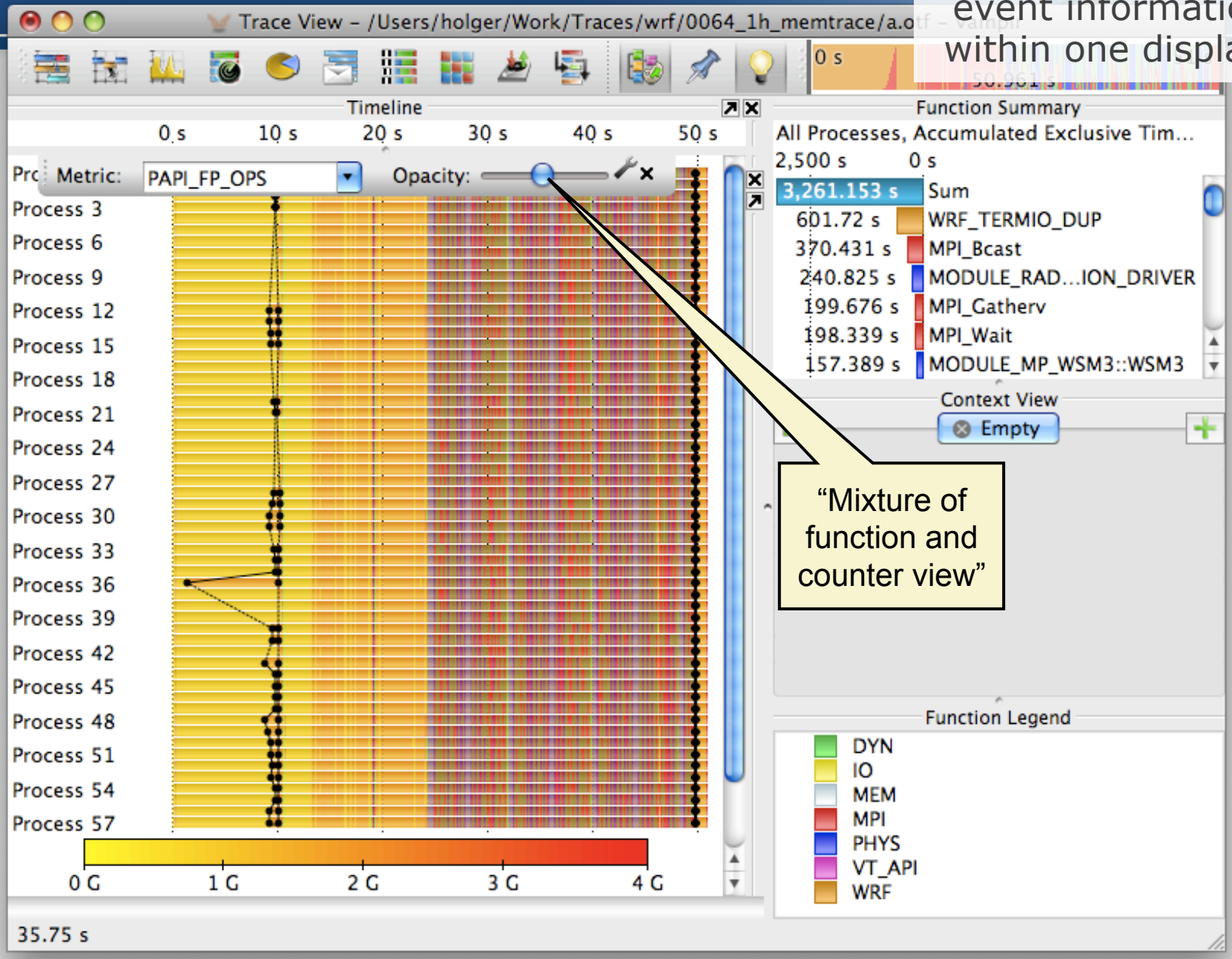
Overlay Functionality of Master Timeline

Correlate counter information with event information within one display.



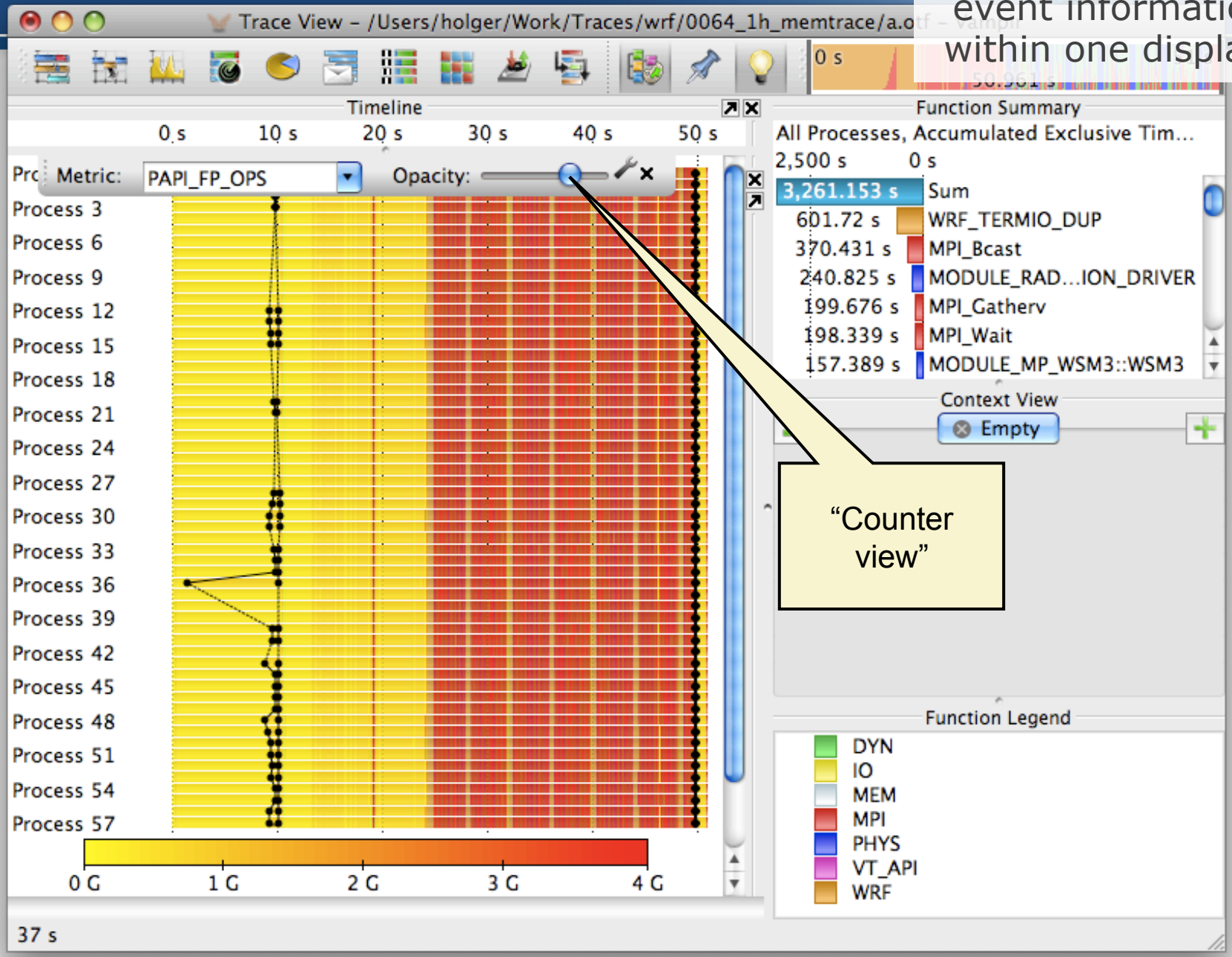
Overlay Functionality of Master Timeline

Correlate counter information with event information within one display.



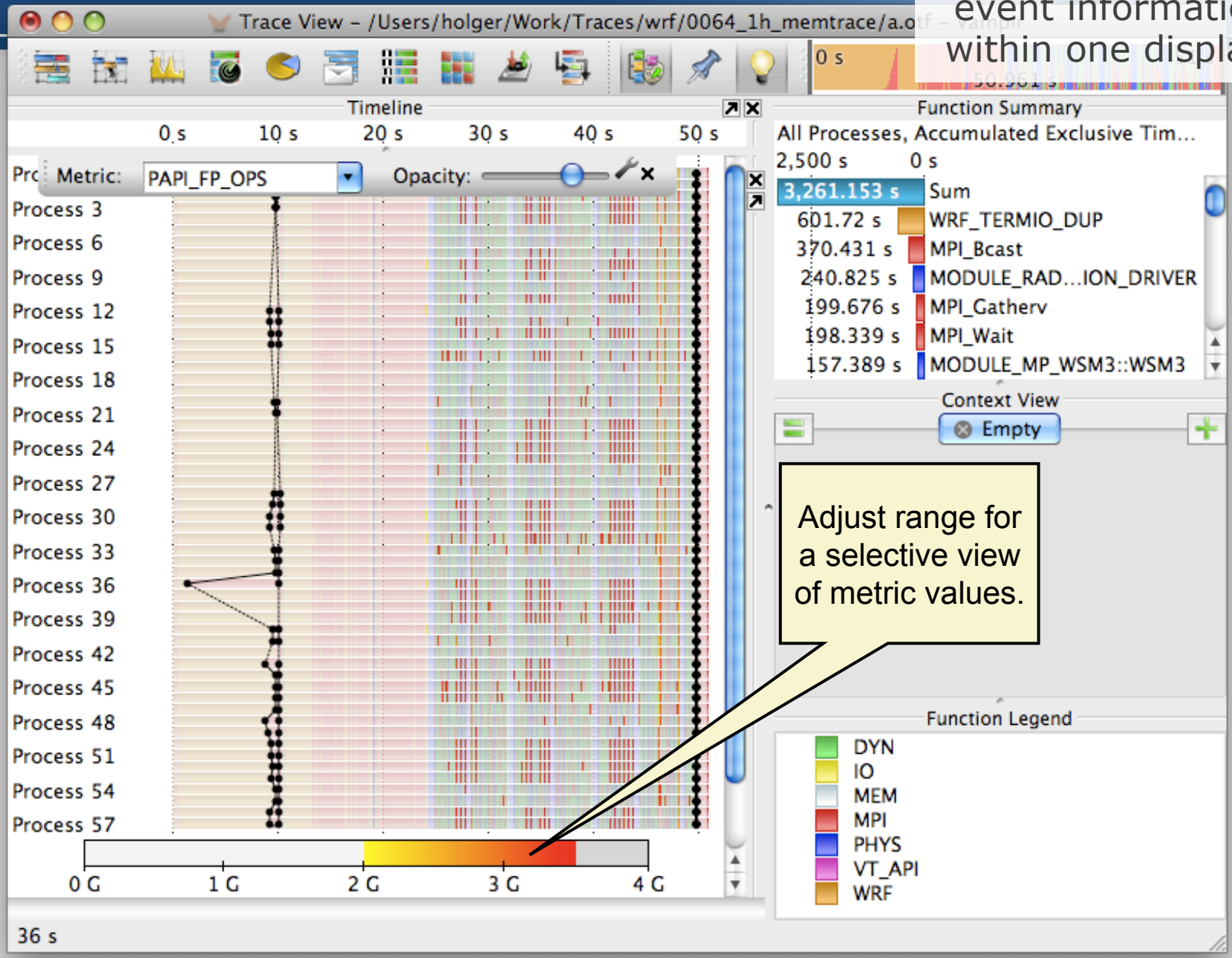
Overlay Functionality of Master Timeline

Correlate counter information with event information within one display.



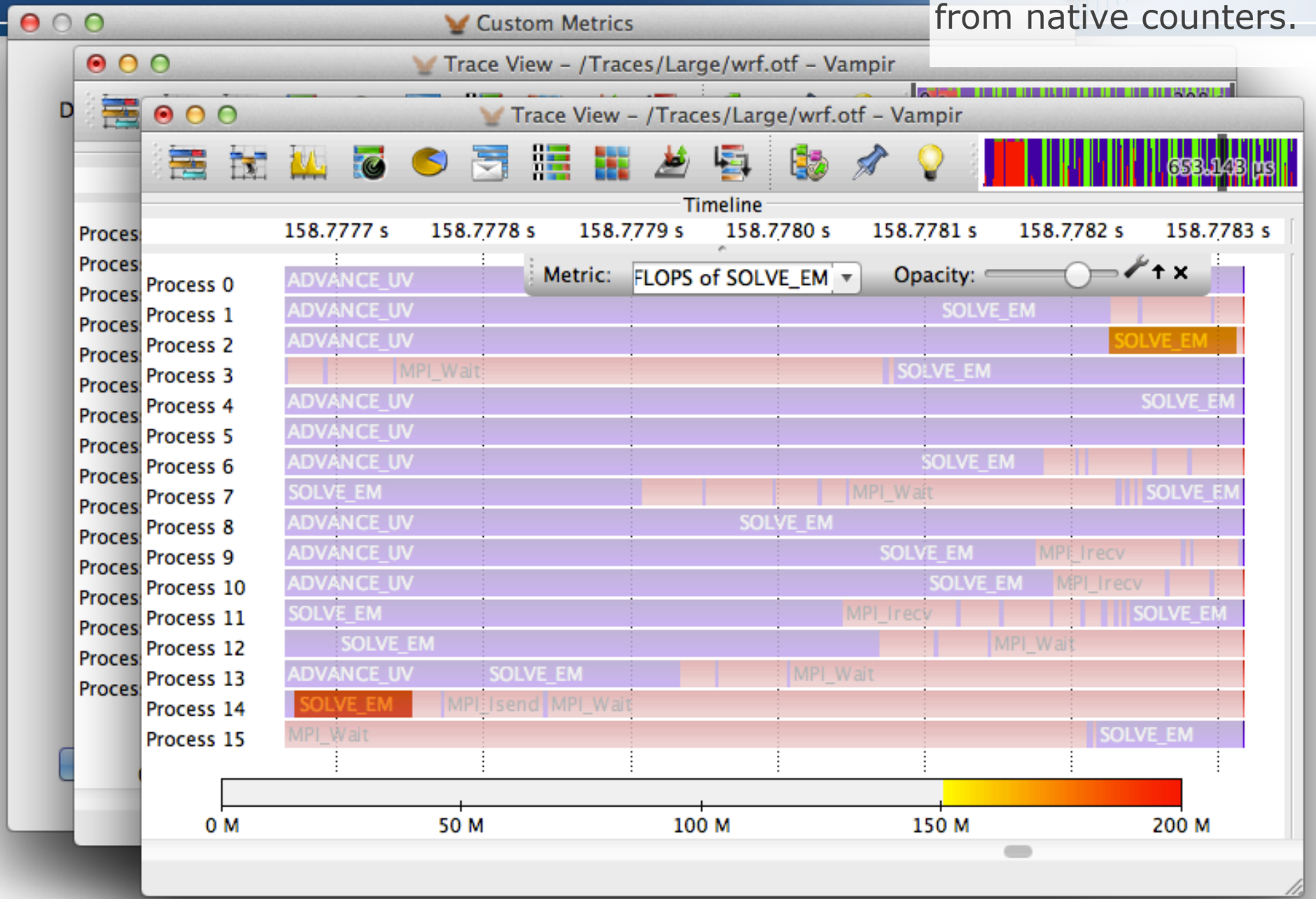
Overlay Functionality of Master Timeline

Correlate counter information with event information within one display.



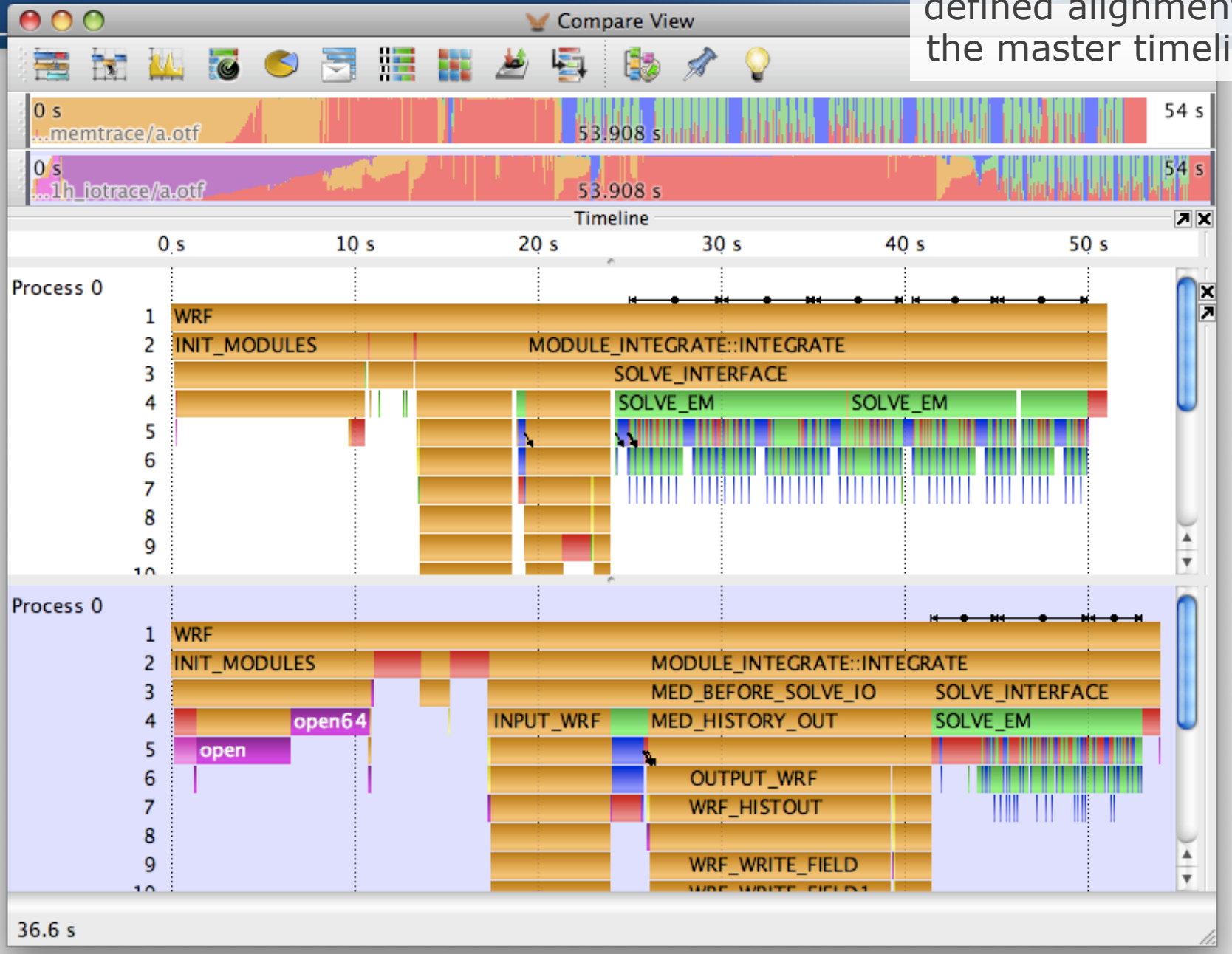
Custom Metrics within the Counter Displays

Create user customized metrics from native counters.



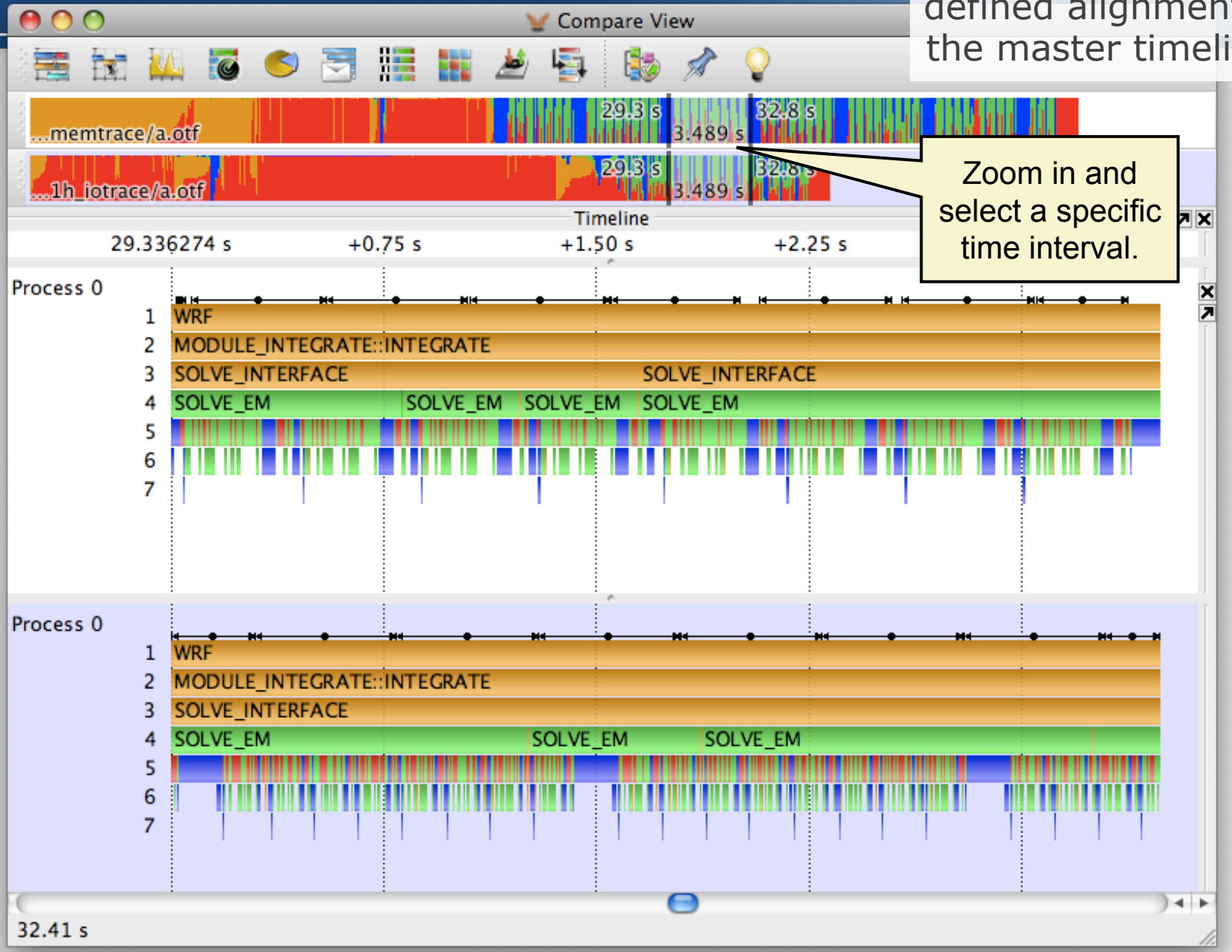
Compare View

Compare multiple trace files by user-defined alignment of the master timeline.



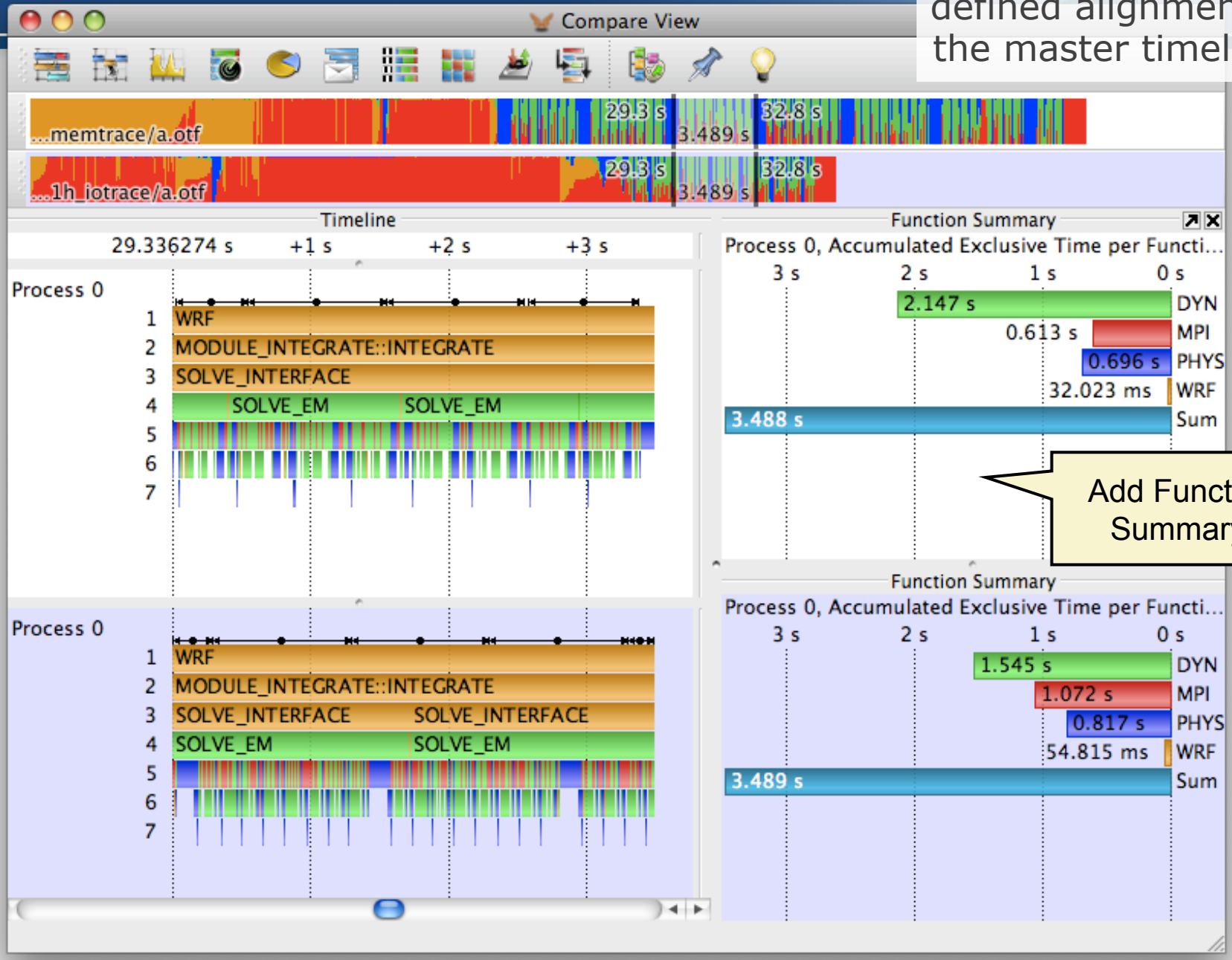
Compare View

Compare multiple trace files by user-defined alignment of the master timeline.



Compare View

Compare multiple trace files by user-defined alignment of the master timeline.



Program Instrumentation

- Detect run-time events (points of interest)
- Pass information to run-time measurement library

Profile Recording

- Collect aggregated information (Time, Counts, …)
- About program and system entities
  - functions, loops, basic blocks
  - application, processes, threads, …

Trace Recording

- Save individual event records together with precise timestamp and process or thread ID
- Plus event specific information

- What do you need to do for it?
  - Use VampirTrace
- Instrumentation (automatic with compiler wrappers)

```
CC=icc
CXX=icpc
F90=ifc
MPICC=mpicc
```

```
CC=vtcc
CXX=vtcxx
F90=vtf90
MPICC=vtcc –vt:mpi -vt:cc mpicc
```

- Re-compile & re-link
- Trace Run (run with appropriate test data set)

- More details later

What does VampirTrace do in the background?

- Instrumentation:
  - Via compiler wrappers
  - By underlying compiler with specific options
  - MPI instrumentation with replacement lib
  - OpenMP instrumentation with Opari
  - Also binary instrumentation with Dyninst
  - Also source2source instrumentation with PDT (Tau)
  - Partial manual instrumentation

What does VampirTrace do in the background?

- Trace Run:
  - Event data collection
  - Precise time measurement
  - Parallel timer synchronization
  - Collecting parallel process/thread traces
  - Collecting performance counters (from PAPI, memory usage, POSIX I/O calls and fork/system/exec calls, and more …)
  - Filtering and grouping of function calls

Summary

- Vampir & VampirServer
  - Interactive trace visualization and analysis
  - Intuitive browsing and zooming
  - Scalable to large trace data sizes (20 TByte)
  - Scalable to high parallelism (20000 processes)
- Vampir for Linux, Windows and MacOS

- VampirTrace
  - Convenient instrumentation and measurement
  - Hides away complicated details
  - Provides many options and switches for experts
- VampirTrace is part of Open MPI since version 1.3

Vampir & VampirTrace

- Event Tracing in General

Common Event Types

- Enter/leave of function/routine/region
  - time stamp, process/thread, function ID
- Send/receive of P2P message (MPI)
  - time stamp, sender, receiver, length, tag, communicator
- Collective communication (MPI)
  - time stamp, process, root, communicator, # bytes
- Hardware performance counter values
  - time stamp, process, counter ID, value
- etc.

- Tracing Advantages
  - Preserve temporal and spatial relationships
  - Allow reconstruction of dynamic behavior on any required abstraction level
  - Profiles can be calculated from traces
- Tracing Disadvantages
  - Traces can become very large
  - May cause perturbation
  - Instrumentation and tracing is complicated
    - Event buffering, clock synchronization, …

- Instrumentation: Process of modifying programs to detect and report events

- There are various ways of instrumentation:
  - Manually
    - Large effort, error prone
    - Difficult to manage
  - Automatically
    - Via source to source translation
    - Via compiler instrumentation
    - Program Database Toolkit (PDT)
    - OpenMP Pragma And Region Instrumenter (Opari)

Open Trace Format (OTF)

- Open source trace file format
- Available at http://www.tu-dresden.de/zih/otf
- Includes powerful libotf for reading/parsing/writing in custom applications
- Multi-level API:
  - High level interface for analysis tools
  - Low level interface for trace libraries
- Actively developed by TU Dresden in cooperation with the University of Oregon and the Lawrence Livermore National Laboratory

- Instrumentation with VampirTrace
  - Hide instrumentation in compiler wrapper
  - Use underlying compiler, add appropriate options

```
CC = mpicc

CC = vtcc –vt:cc mpicc
```

- Test Run
  - User representative test input
  - Set parameters, environment variables, etc.
  - Perform trace run
- Get Trace

Source Code Instrumentation

```
int foo(void* arg) {

    if (cond) {

        return 1;
    }

    return 0;
}
```

```
int foo(void* arg) {
    enter(7);
    if (cond) {
        leave(7);
        return 1;
    }
    leave(7);
    return 0;
}
```

manually or automatically