# Performance Analysis with Vampir

Matthias Weber
Technische Universität Dresden

VAMPIR

# Outline

- **Part I: Performance Analysis with Vampir & Score-P**
  - Introduction/Performance Engineering
  - Score-P – Measurement System
  - Vampir – Performance Data Visualization
- **Part II: Vampir Analysis Exercise**
  - Analysing Four Application Traces
- **Part III: Vampir Hands-On**
  - Visualizing and Analyzing NPB-MZ-MPI / BT

# Introduction
# Performance Engineering

# Disclaimer

It is extremely easy to waste performance!

- Bad MPI (50-90%)
- No node-level parallelism (94%)
- No vectorization (75%)
- Bad memory access pattern (99%)
- Total: **1/2** x **1/16** x **1/4** * **1/100**
- ~**1/10000** of the peak performance

# Disclaimer (2)

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

# Virtual Institute – High Productivity Supercomputing

- **Goal**: Improve the quality and accelerate the development process of complex simulation codes running on highly-parallel computer systems
- Start-up funding (2006–2011)
  by Helmholtz Association of German Research Centres
- Activities
  - Development and integration of HPC programming tools
    - Correctness checking & performance analysis
  - Academic workshops
  - Training workshops
  - Service
    - Support email lists
    - Application engagement

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES

**http://www.vi-hps.org**

# VI-HPS partners (founders)

## Forschungszentrum Jülich

- Jülich Supercomputing Centre

## RWTH Aachen University

- Centre for Computing & Communication

## Technische Universität Dresden

- Centre for Information Services & HPC

## University of Tennessee (Knoxville)

- Innovative Computing Laboratory

# VI-HPS partners (cont.)

Arm Ltd.
- Allinea Software

Barcelona Supercomputing Center
- Centro Nacional de Supercomputación

Lawrence Livermore National Lab.
- Center for Applied Scientific Computing

Leibniz Supercomputing Centre

Technical University of Darmstadt
- Laboratory for Parallel Programming

allinea

Barcelona Supercomputing Center Centro Nacional de Supercomputación

Lawrence Livermore National Laboratory

lrz Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften

TECHNISCHE UNIVERSITÄT DARMSTADT

# VI-HPS partners (cont.)

## Technical University of Munich
- Chair for Computer Architecture

## University of Oregon
- Performance Research Laboratory

## University of Stuttgart
- HPC Centre

## University of Versailles St-Quentin
- LRC ITACA

TECHNISCHE UNIVERSITÄT MÜNCHEN

UNIVERSITY OF OREGON

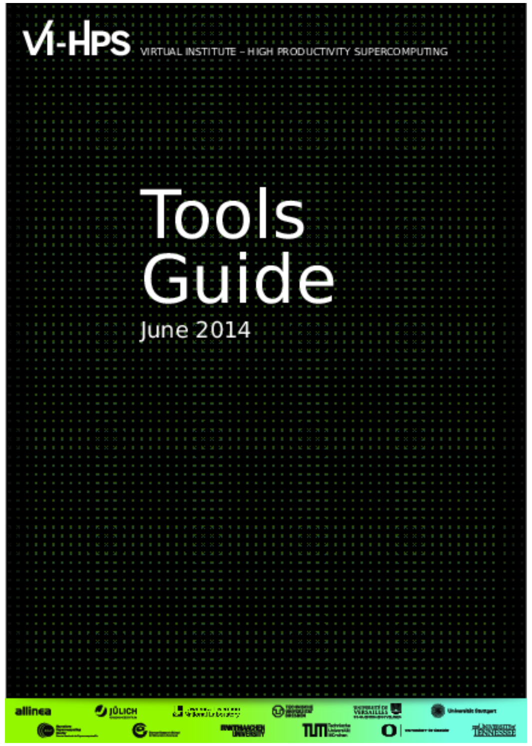
Universität Stuttgart

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

# Productivity tools

- **MUST & Archer**
  - MPI & OpenMP usage correctness checking
- PAPI
  - Interfacing to hardware performance counters
- Periscope Tuning Framework
  - Automatic analysis and Tuning
- **Scalasca**
  - Large-scale parallel performance analysis
- **TAU**
  - Integrated parallel performance system
- **Vampir**
  - Interactive graphical trace visualization & analysis
- **Score-P**
  - Community-developed instrumentation & measurement infrastructure

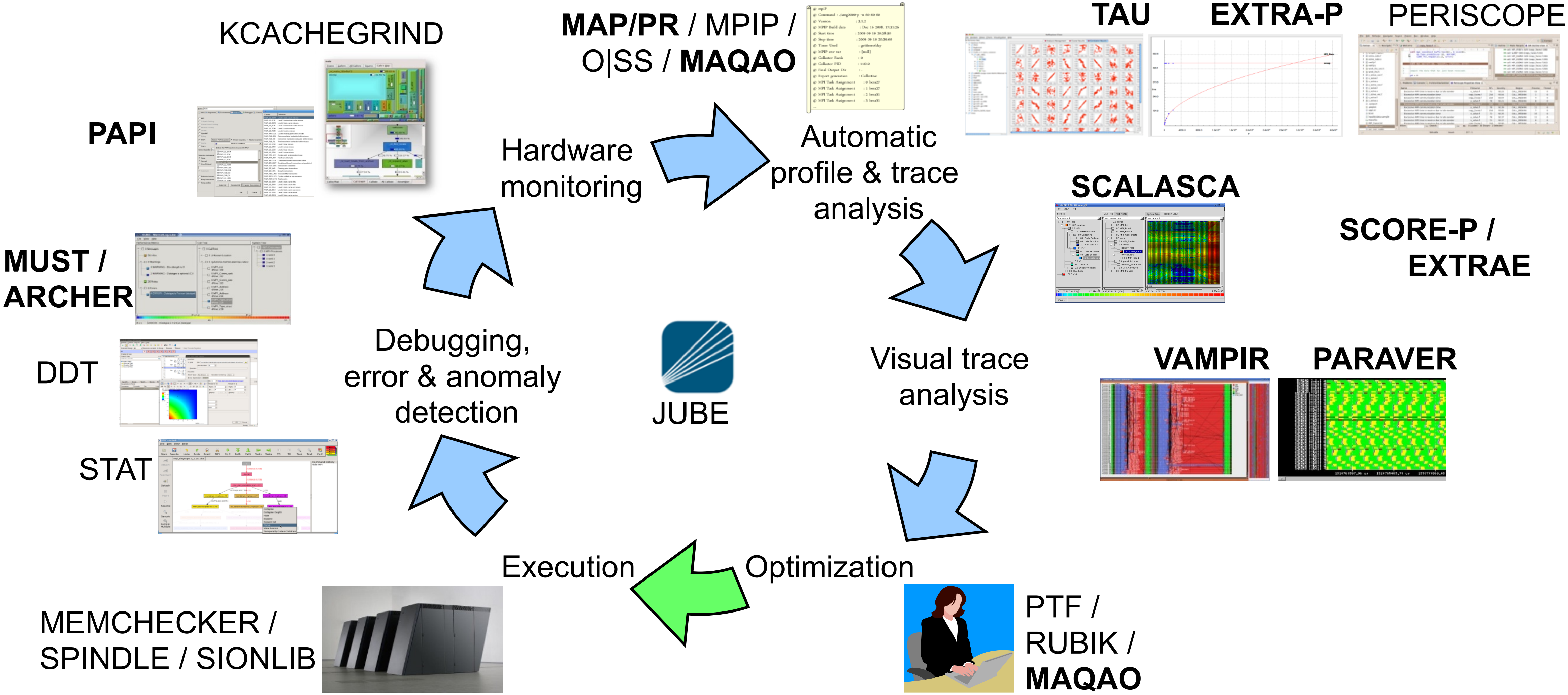
See VI-HPS Tools Guide for a overview of tools: https://www.vi-hps.org/tools



# Productivity tools (cont.)

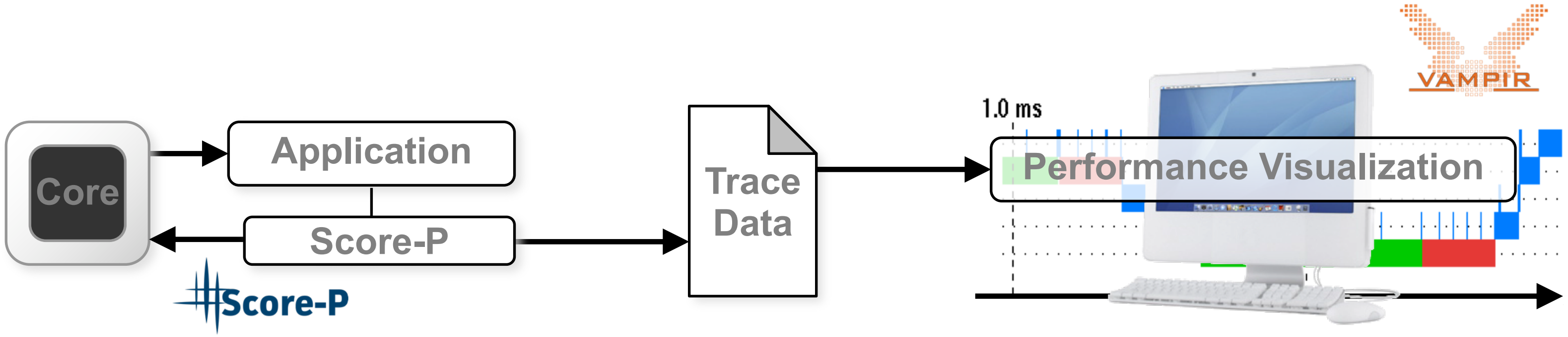
- FORGE DDT/**MAP/PR**: Parallel debugging, profiling & performance reports
- **Extra-P**: Automated performance modelling
- Kcachegrind: Callgraph-based cache analysis [x86 only]
- JUBE: Workflow execution environment
- **MAQAO**: Assembly instrumentation & optimization [x86-64 only]
- mpiP/mpiPview: MPI profiling tool and analysis viewer
- Open MPI: Integrated memory checking
- Open|SpeedShop: Integrated parallel performance analysis environment
- **Paraver/Dimemas/Extrae**: Event tracing, graphical trace visualization & analysis
- Rubik: Process mapping generation & optimization [BG only]
- SIONlib/Spindle: Optimized native parallel file I/O & shared library loading
- STAT: Stack trace analysis tools

# Technologies and their integration

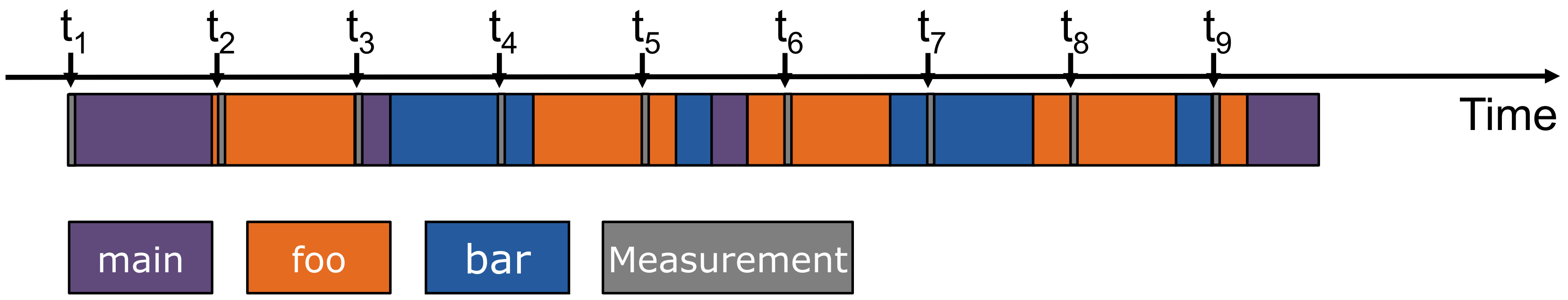


# Recording and Studying a Program’s Behavior

- Performance analysis in practice:
  - Attach measurement system Score-P to application
  - Run application with Score-P monitor
    - Result: trace/profile data
- Study the trace data with Vampir
- Repeat
  - Adapt instrumentation (“what you measure”)
  - Evaluate result of a change

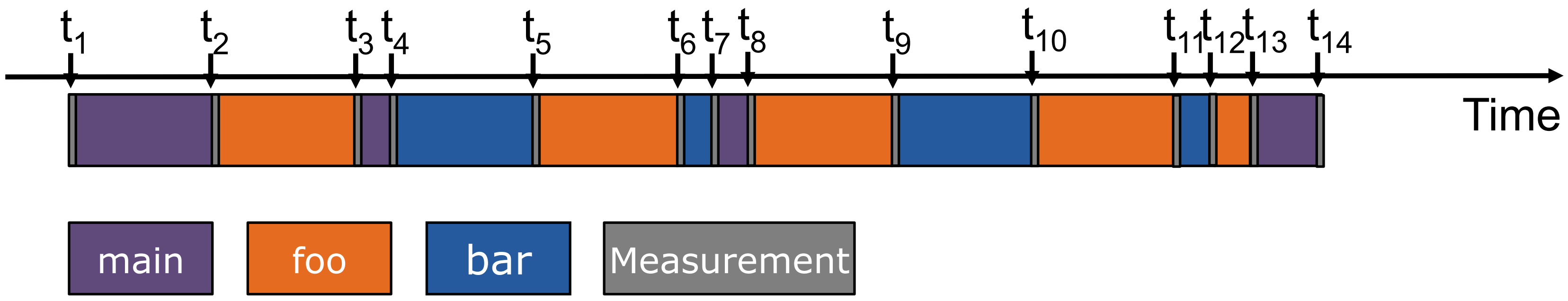


# Methodology: Sampling




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

# Methodology: Instrumentation




- 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

# Instrumentation vs. Sampling

- Instrumentation/Tracing:
  - Typically more difficult than using a profiler
  - Does not garuantee anything about overhead or recording size
    - **It depends on the event rate**
      - E.g. an MPI-only trace has very low overhead
    - Filtering required

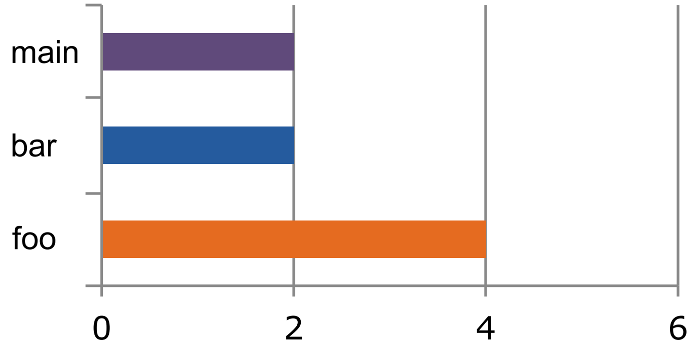
- Sampling:
  - Does not give an absolutely accurate picture of a run
  - Cannot count function calls
  - Cannot record exact timings
  - Cannot record exact performance counters
  - It is **statistical sampling**
  - It cannot capture semantics of APIs, i.e. it cannot follow API usage and analyze passed arguments, e.g. transferred bytes

# Methodology: Profiling vs. Tracing

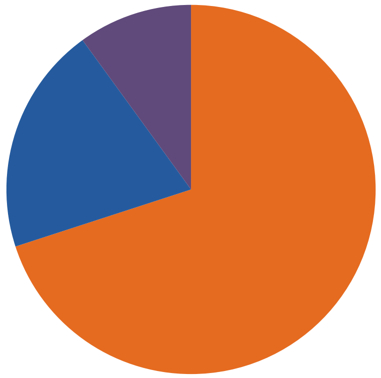
Score-P

- Statistics

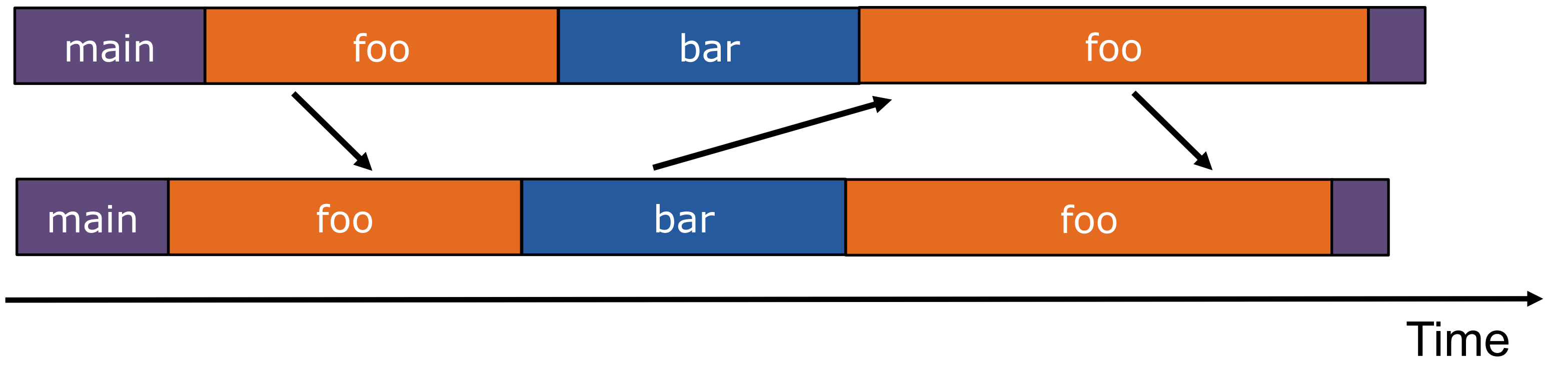
Number of Invocations



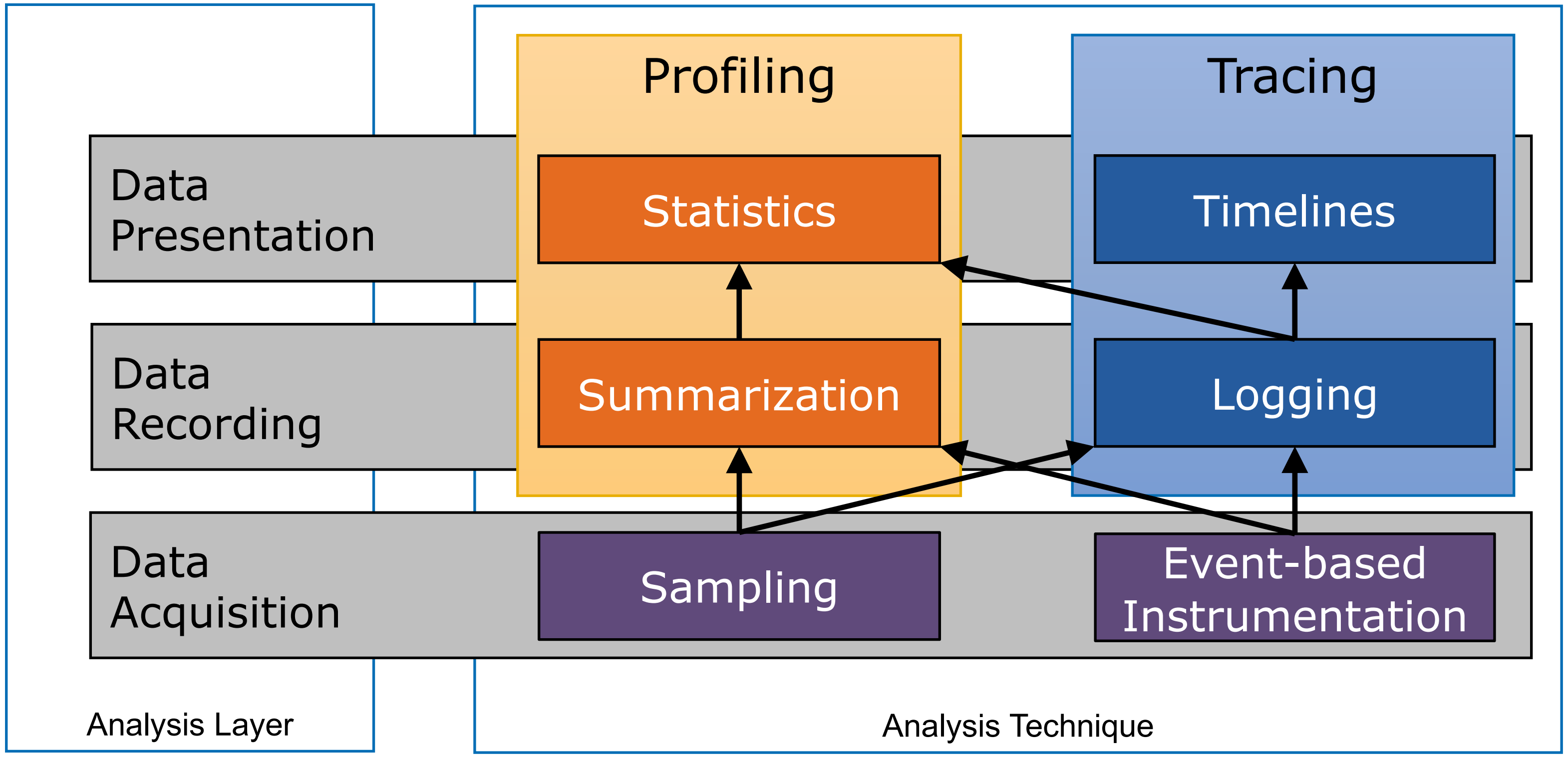

Execution Time



- Timelines



# Terms Used and How They Connect



# Score-P Measurement System

# Fragmentation of tools landscape

- Several performance tools co-exist
  - Separate measurement systems and output formats
- Complementary features and overlapping functionality
- Redundant effort for development and maintenance
  - Limited or expensive interoperability
- Complications for user experience, support, training

| | | | |
|---|---|---|---|
| Vampir | Scalasca | TAU | Periscope |
| VampirTrace OTF | EPILOG / CUBE | TAU native formats | Online measurement |

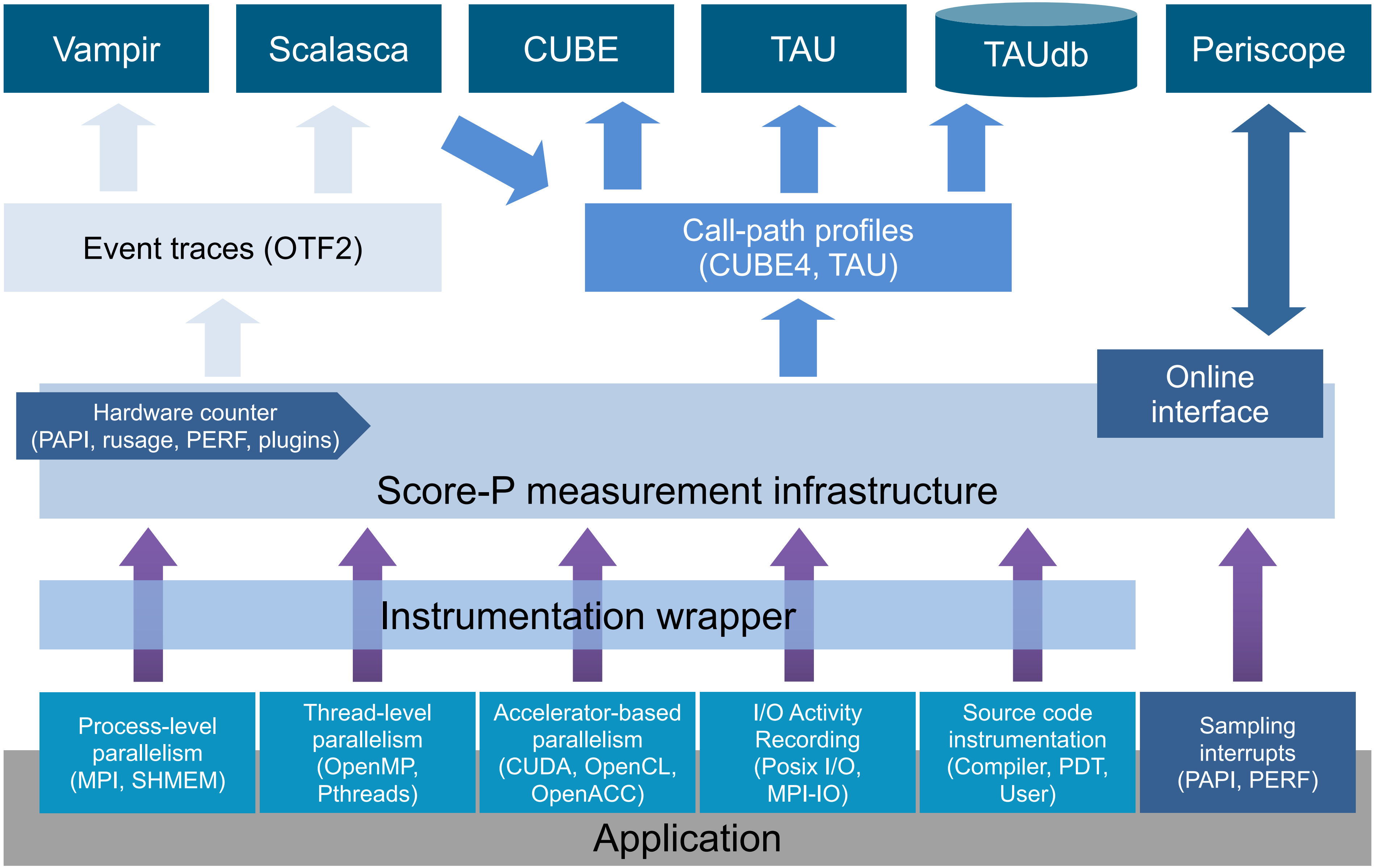
# Score-P project idea

- Start a community effort for a common infrastructure
  - Score-P instrumentation and measurement system
  - Common data formats OTF2 and CUBE4
- Partners:
  - TU Dresden, FZ Jülich, TU München, University of Oregon, RWTH Aachen, TU Darmstadt

- Developers save manpower by sharing development resources
- Users need only a single installation, resulting in single learning curve

- Ensure a single official release version at all times which will always work with the tools
- Commitment to joint long-term cooperation
  - Development based on meritocratic governance model
  - Open for contributions and new partners

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung


Vampir
Scalasca
CUBE
TAU
TAUdb
Periscope
Event traces (OTF2)
Call-path profiles (CUBE4, TAU)
Online interface
Hardware counter (PAPI, rusage, PERF, plugins)
Score-P measurement infrastructure
Instrumentation wrapper
Process-level parallelism (MPI, SHMEM)
Thread-level parallelism (OpenMP, Pthreads)
Accelerator-based parallelism (CUDA, OpenCL, OpenACC)
I/O Activity Recording (Posix I/O, MPI-IO)
Source code instrumentation (Compiler, PDT, User)
Sampling interrupts (PAPI, PERF)
Application

# Data Collection with Score-P

Score-P

- Instrument (Compile & Link)

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

```
CC    = scorep <options> icc
CXX   = scorep <options> icpc
F90   = scorep <options> ifc
MPICC = scorep <options> mpicc
```

- Execute

```
$ ./a.out
```

```
$ ./mpirun -np 2 ./a.out
```

- Inspect

```
$ ls -R
scorep-20170323_1309_7243761919249966 a.out

./scorep-20170323_1309_7243761919249966:
profile.cubex scorep.cfg
```

# Data Collection with Score-P

Score-P

- 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
   [...]
```

# Data Collection with Score-P

- Example for generating and loading a trace:

```
$ export SCOREP_ENABLE_PROFILING=false
$ export SCOREP_ENABLE_TRACING=true
$ export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC

$ mpirun -np 4 ./a.out

$ ls -R
scorep-20170323_1309_7243761919249966 a.out

./scorep-20170323_1309_7243761919249966:
scorep.cfg traces/ traces.def traces.otf2

$ vampir scorep-20170323_1309_7243761919249966/traces.otf2
```

# Data Collection with Score-P

- Score-P supports a combination of instrumentation/sampling:
  - Instrumentation for MPI/OpenMP events
  - Sampling for everything else

- Simple configuration, e.g.:

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

# 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
```

- Filter file:

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

# Good Tracing Practices

Score-P

- Traces can become large; Their handling challenging
  - Trace size proportional to: number of processes/threads (width), duration (length), and measurement detail (depth)

- Intermediate flushes => High degree of perturbation
  - Either use less detail or larger trace buffers

- Traces should be written to a parallel file systems
  - E.g. "/work", "/scratch", or "/p/lscratchc"

- Moving large traces can become challenging
  - However, systems with more memory/core can analyze larger traces
  - Alternatively, undersubscribe for increased memory/core ratios

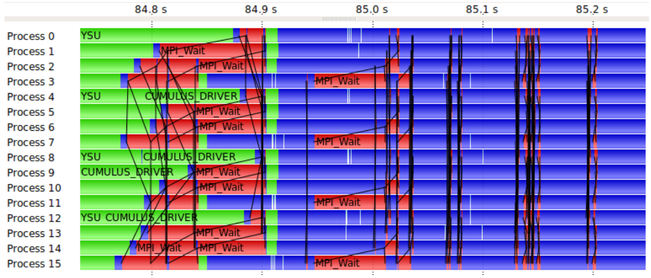
# Vampir
# Performance Data Visualization

# Event Trace Visualization with Vampir

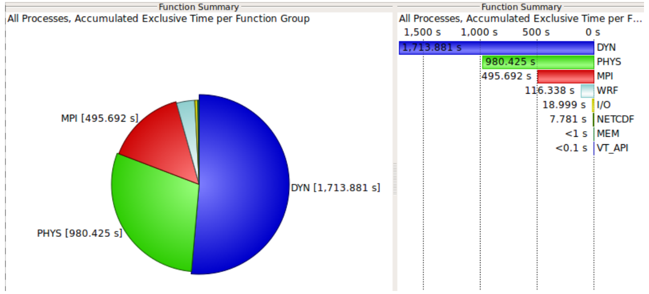
- Visualization of dynamic runtime behaviour at any level of detail along with statistics and performance metrics
- Alternative and supplement to automatic analysis

- **Typical questions that Vampir helps to answer**
  - What happens in my application execution during a given time in a given process or thread?
  - How do the communication patterns of my application execute on a real system?
  - Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?

- **Timeline charts**
  - Application activities and com-munication along a time axis



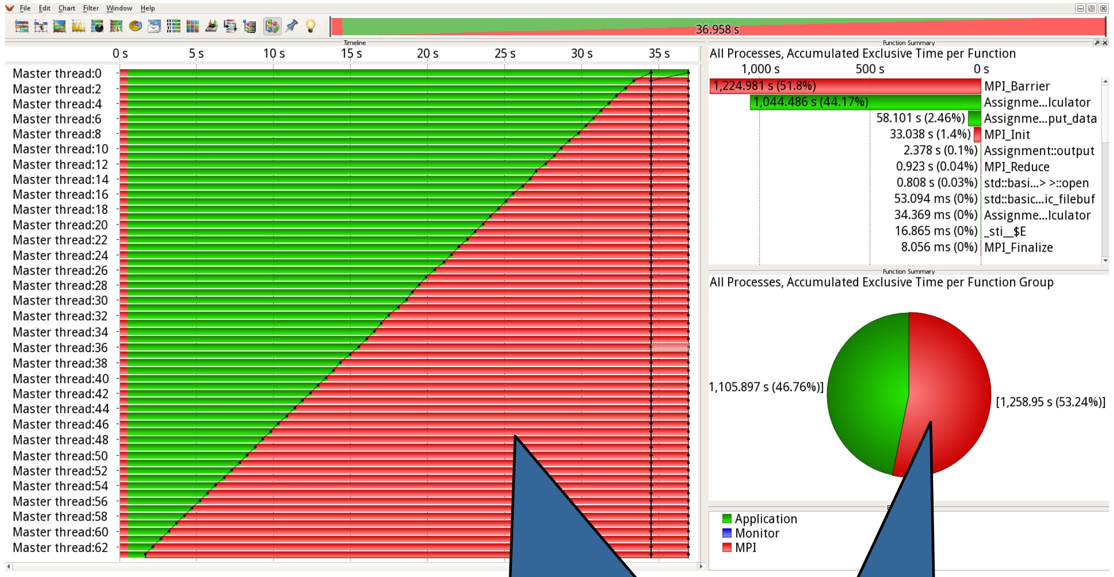

- **Summary charts**
  - Quantitative results for the currently selected time interval



# Event Trace Visualization with Vampir
## The value of seeing how an application executes on the machine

- Application code computing coulomb forces
- The workload was distributed evenly across available processes
- The user expected perfect parallelized code
- However the underlying algorithm worked differently than expected

Visualization of the application execution instantly shows a problem in the parallelization approach




Large imbalance instantly visible

More than 50% application time wasted!

# Main Performance Charts of Vampir

## Timeline Charts

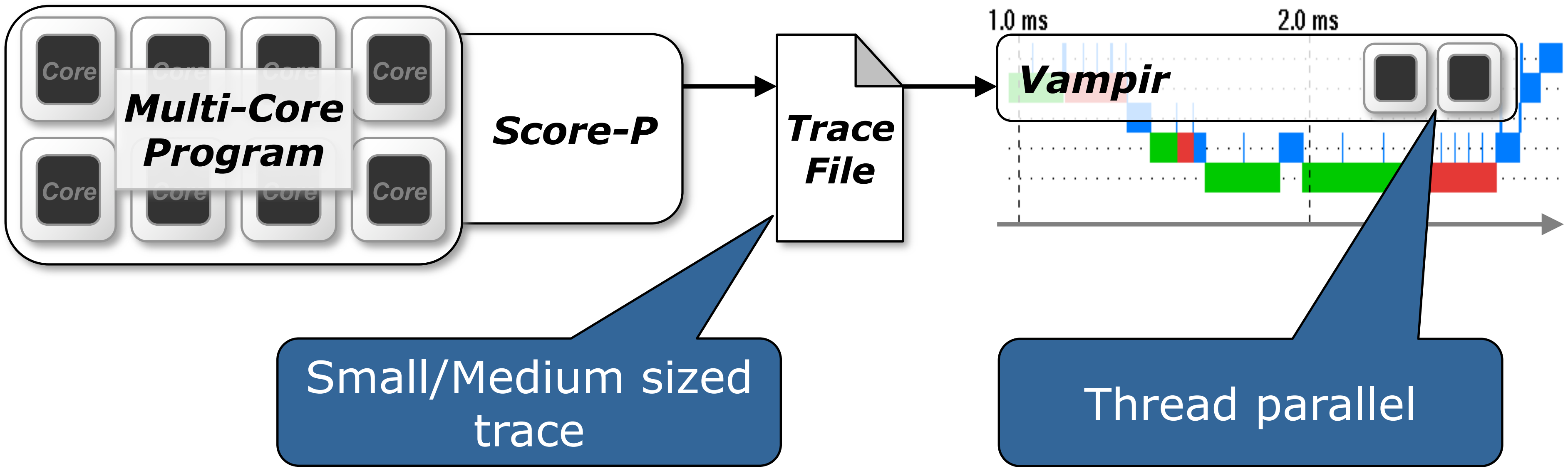
| | | |
|---|---|---|
| Master Timeline | ➡ | *all threads' activities* |
| Process Timeline | ➡ | *single thread's activities* |
| Summary Timeline | ➡ | *all threads' function call statistics* |
| Performance Radar | ➡ | *all threads' performance metrics* |
| Counter Data Timeline | ➡ | *single threads' performance metrics* |
| I/O Timeline | ➡ | *all threads' I/O activities* |

## Summary Charts

| | |
|---|---|
| Function Summary | Process Summary |
| Message Summary | Communication Matrix View |
| I/O Summary | Call Tree |

# Visualization Modes (1)
## Directly on front end or local machine

```
% vampir
```

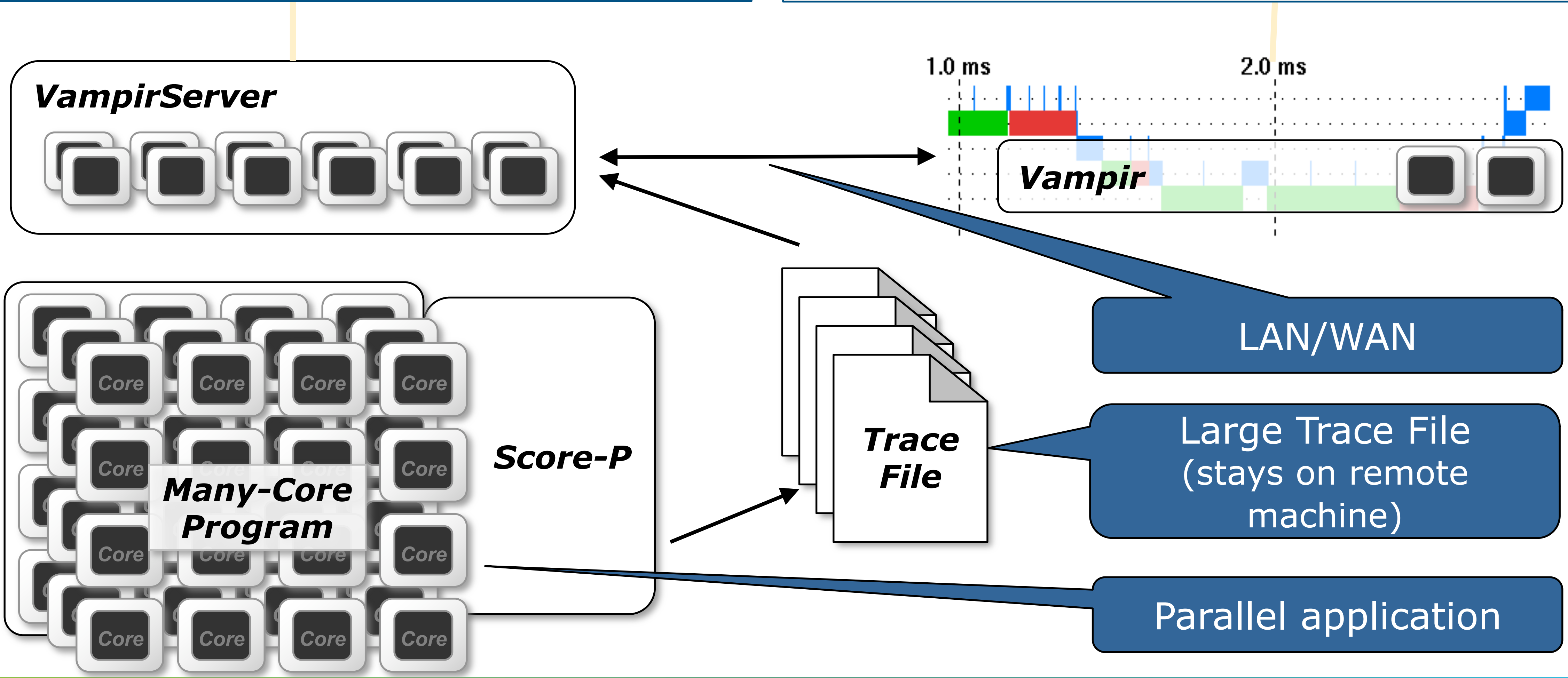


# Visualization Modes (2)

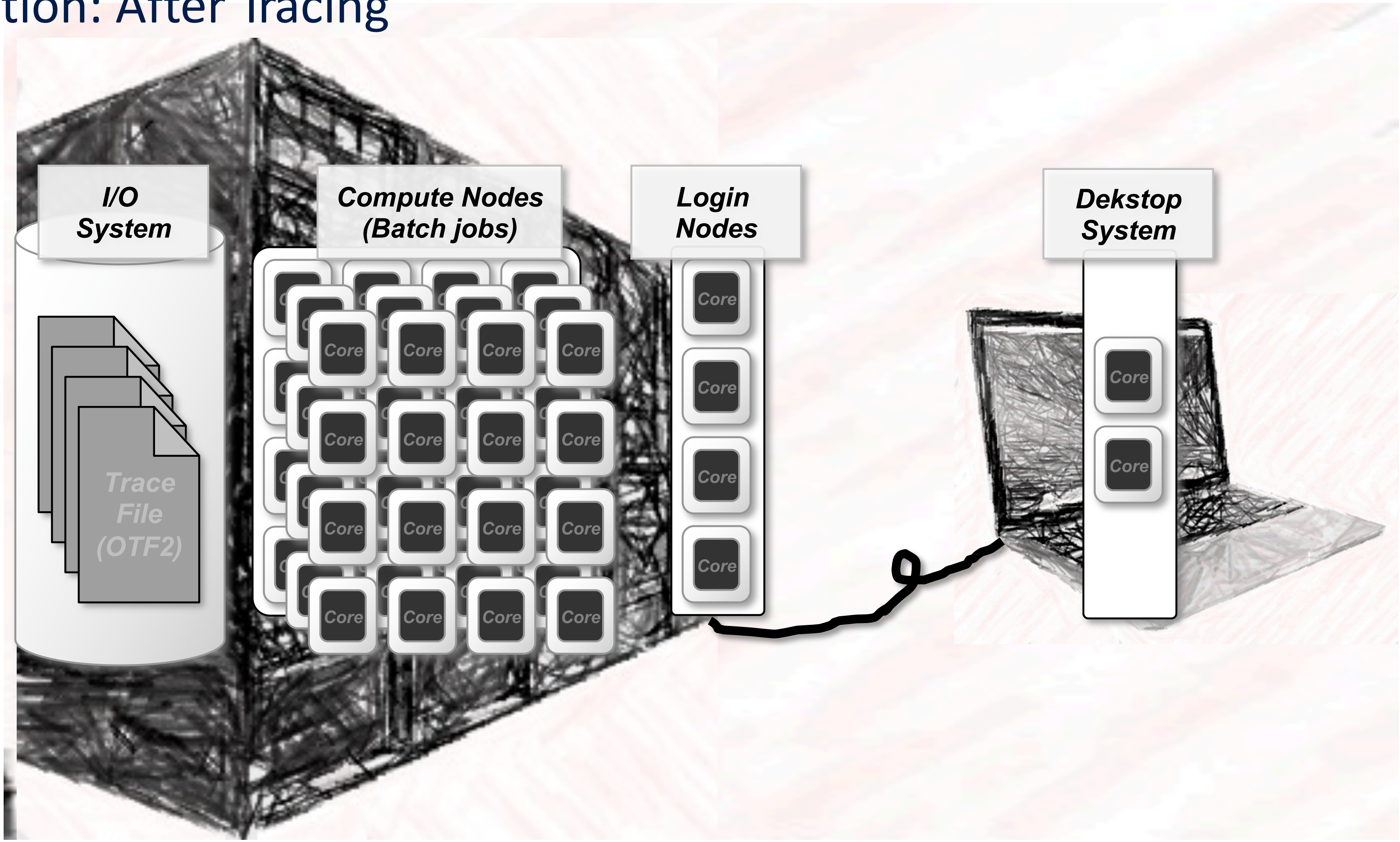
## On local machine with remote VampirServer

```
% vampirserver start
```

```
% vampir
```

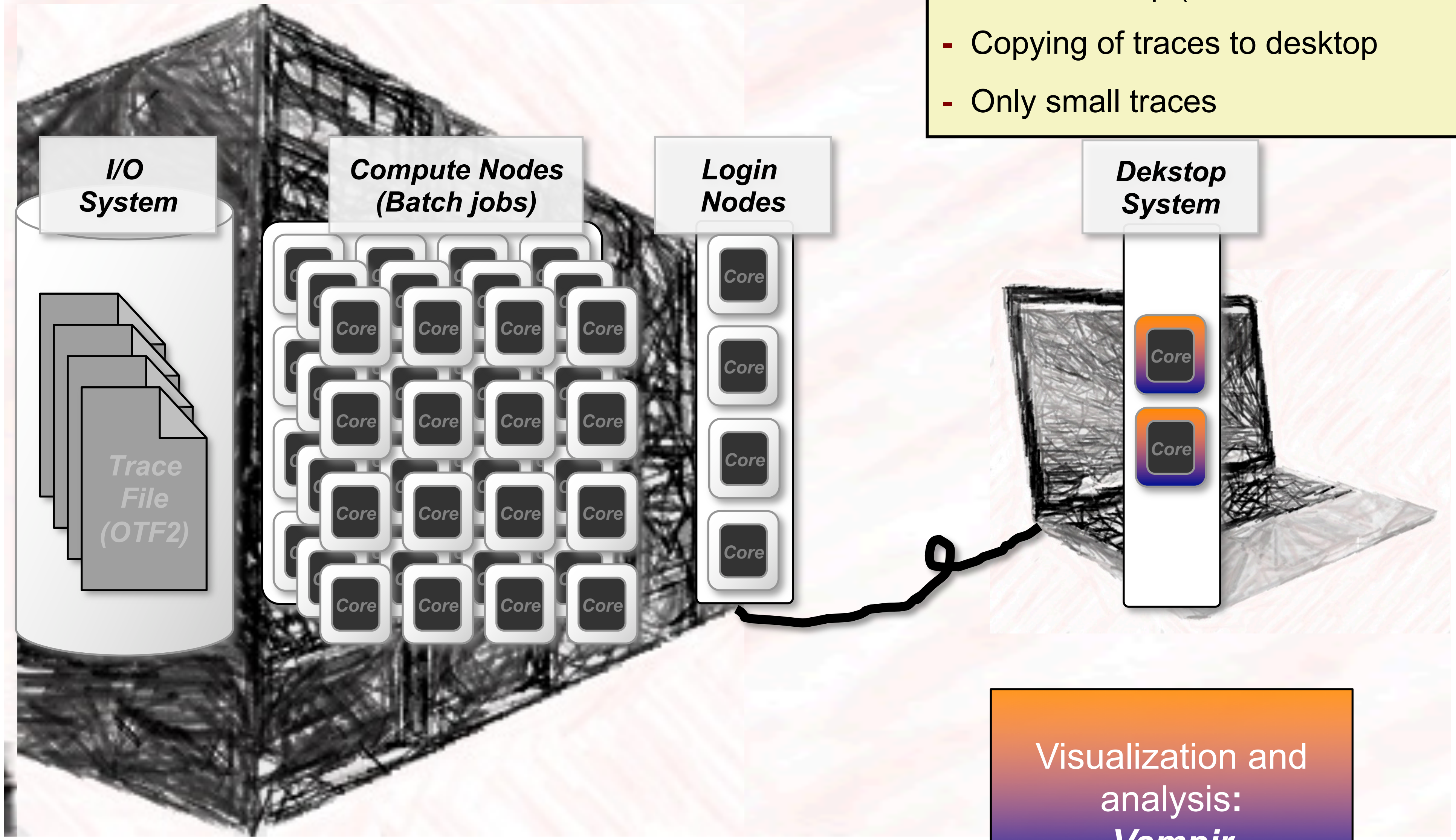


# Visualization: After Tracing



# Visualization: Most simple (Analysis on Desktop)

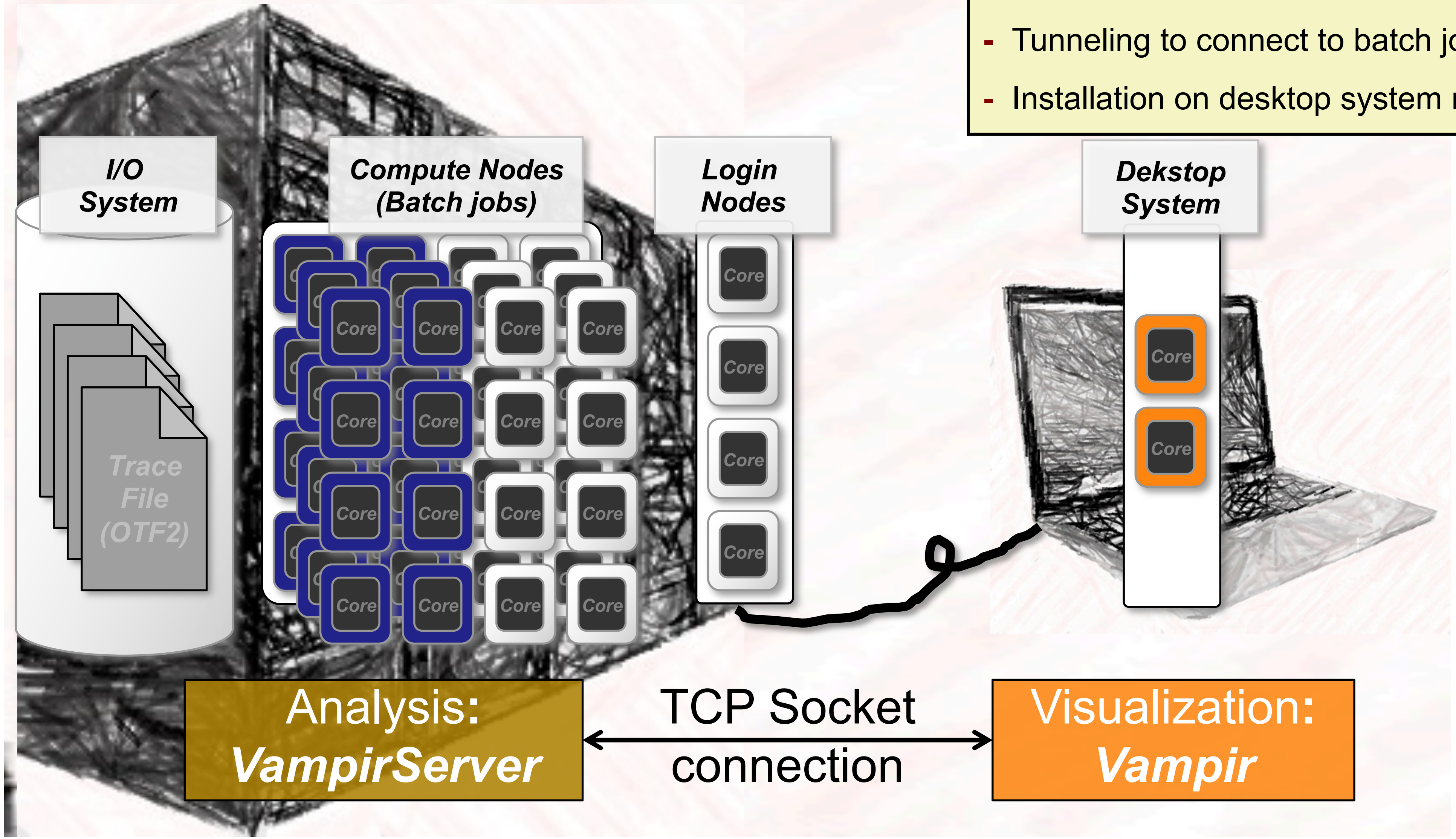
+ Minimal setup (no installations, no batch job)
- Copying of traces to desktop
- Only small traces




Visualization and analysis:
***Vampir***

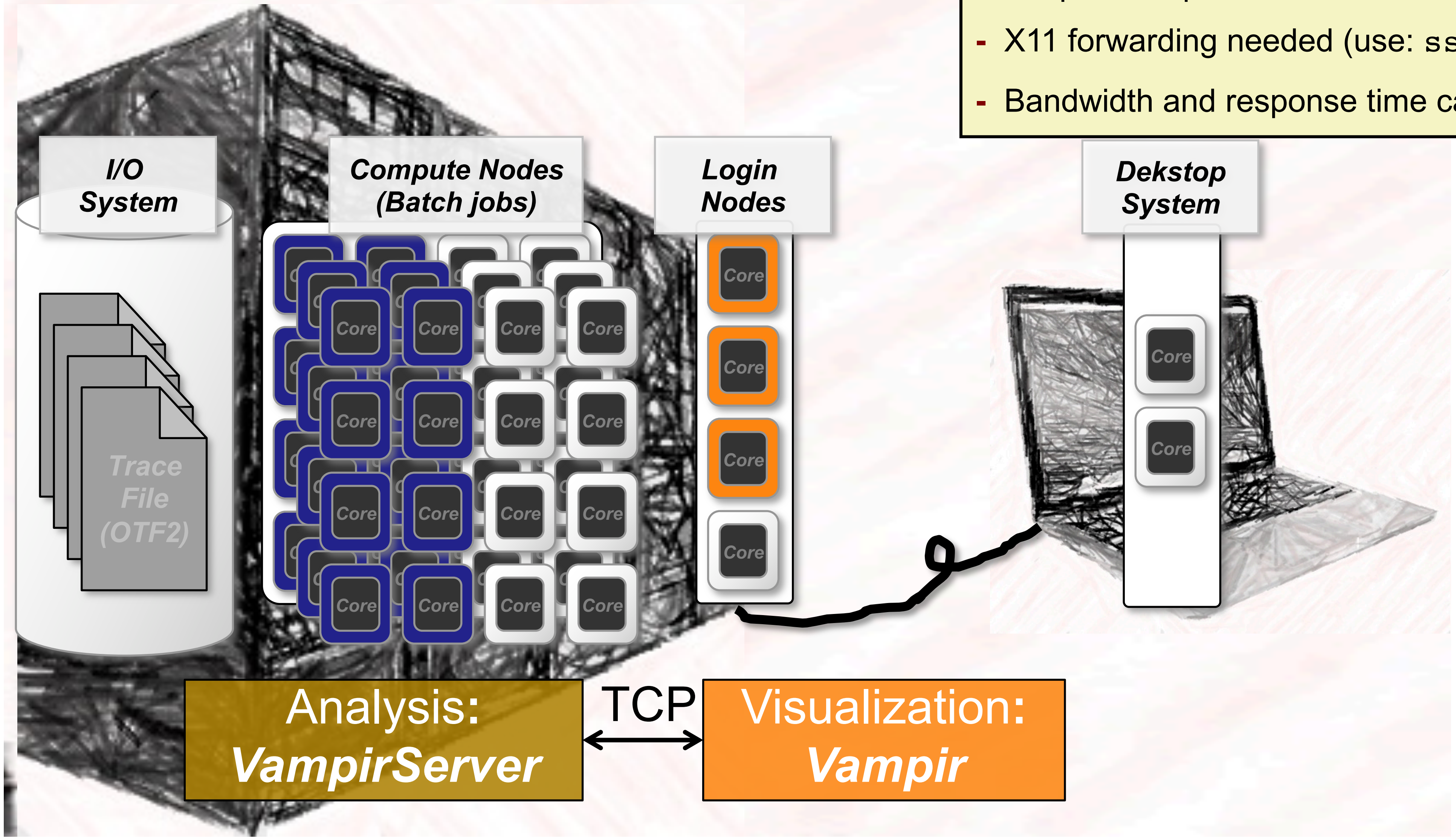
# Visualization: Best Option (Analysis on HPC system)

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



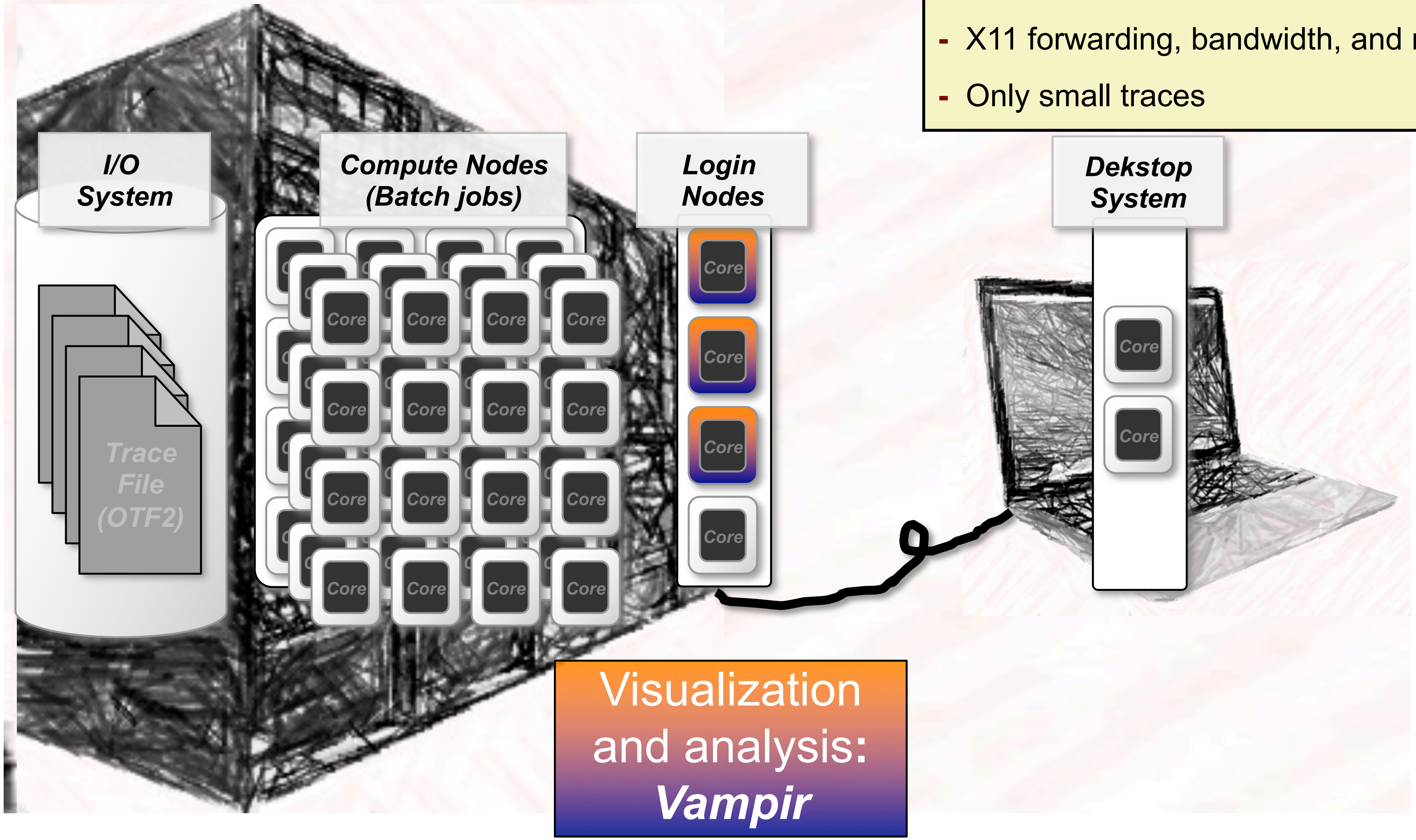
# Visualization: Alternative (Analysis on HPC system)

+ Simpler setup, no installation on desktop
- X11 forwarding needed (use: `ssh –XC …`)
- Bandwidth and response time can be critical

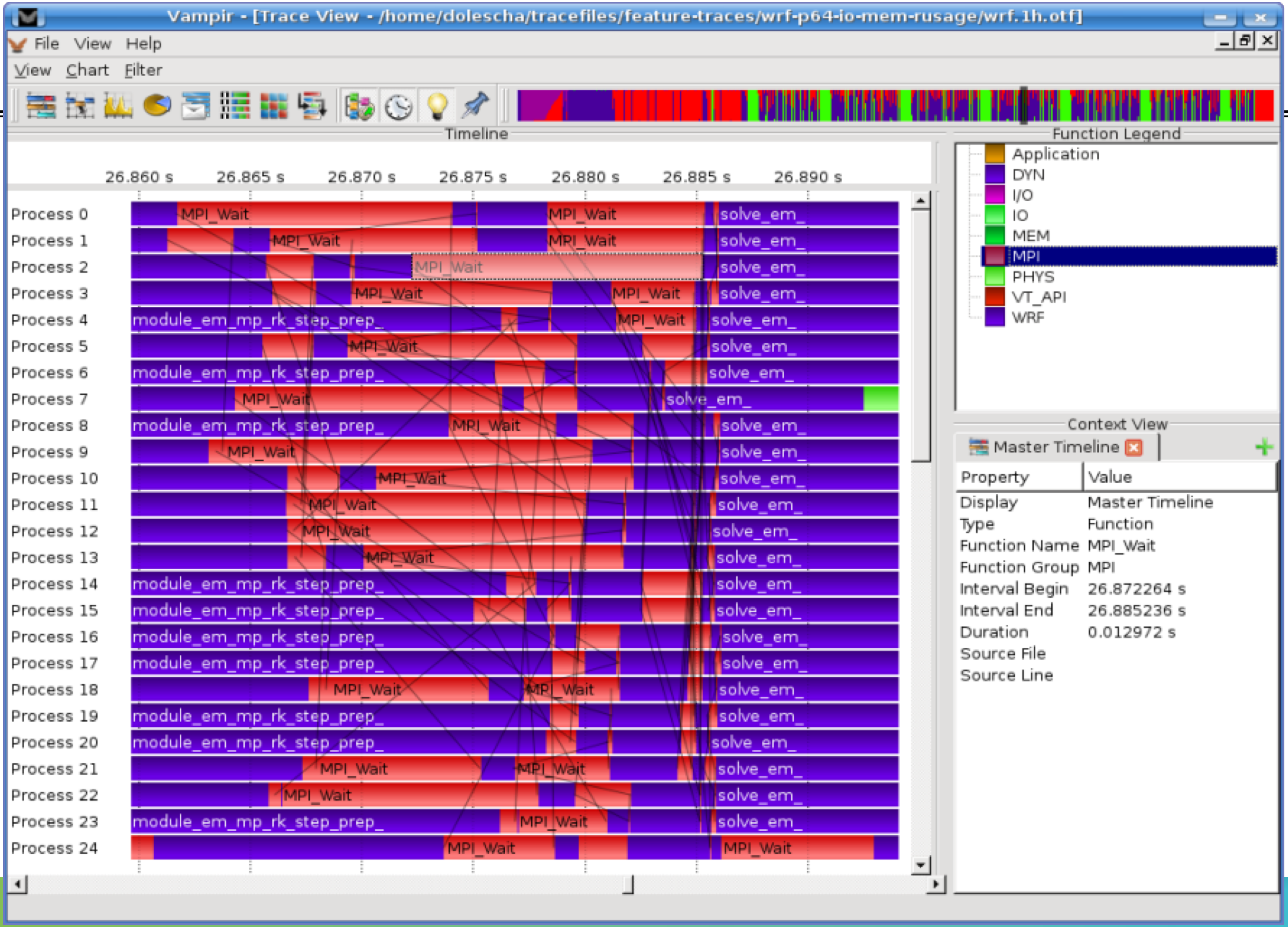


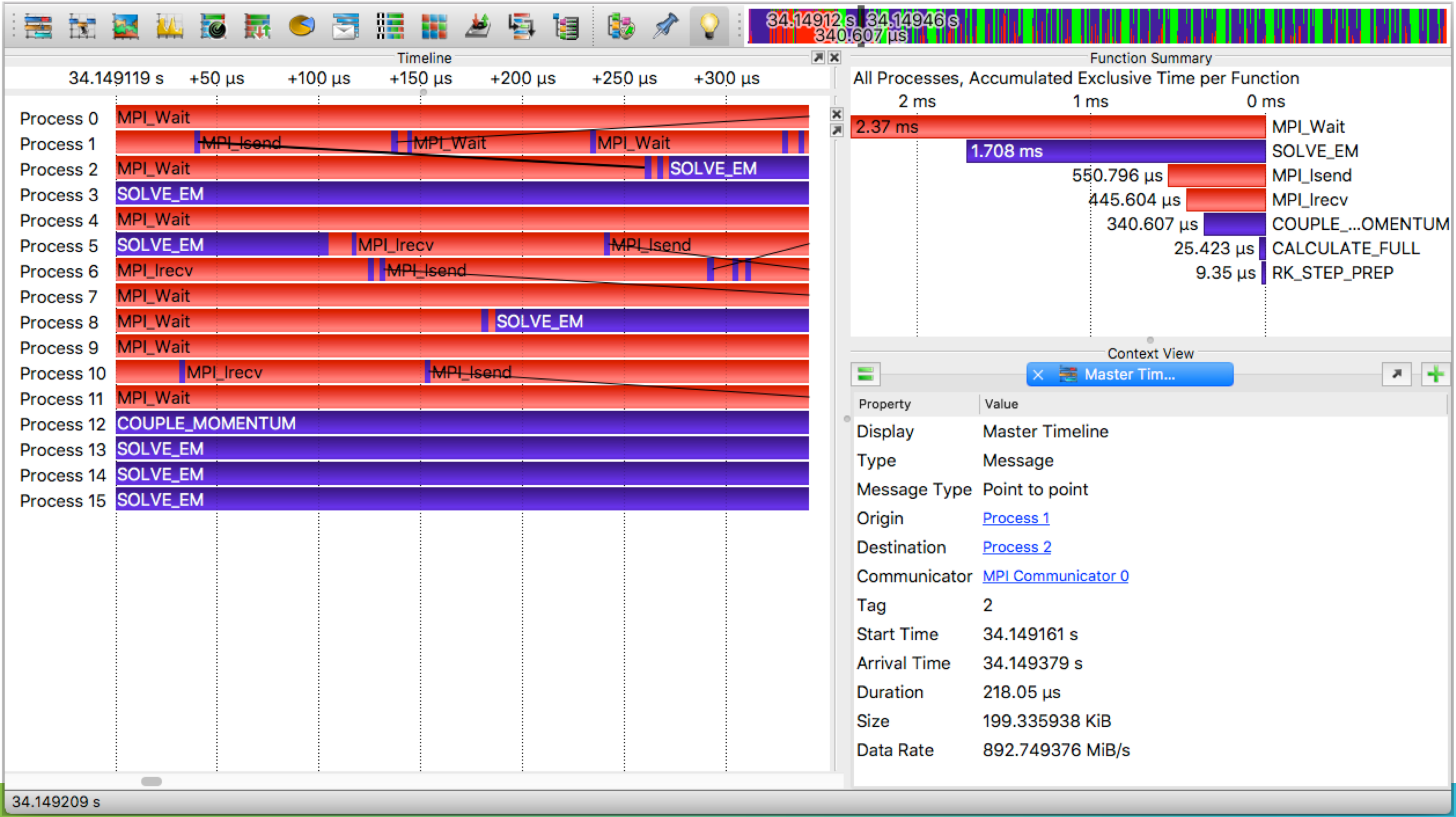
# Visualization: Most simple (Analysis on Frontend)

+ Minimal setup (no installations, no batch job)
- X11 forwarding, bandwidth, and response
- Only small traces

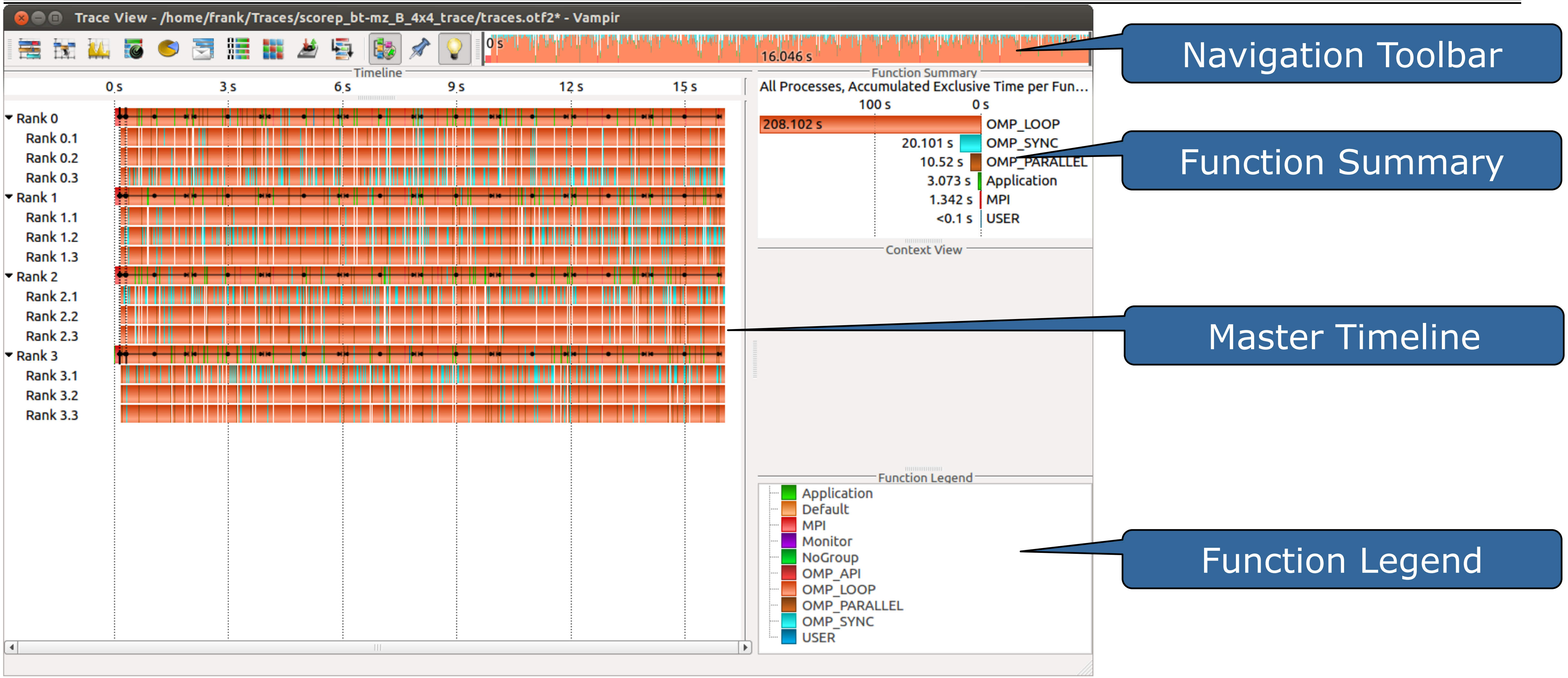


# Example Analysis:
# Visualizing and Analyzing NPB-MZ-MPI / BT

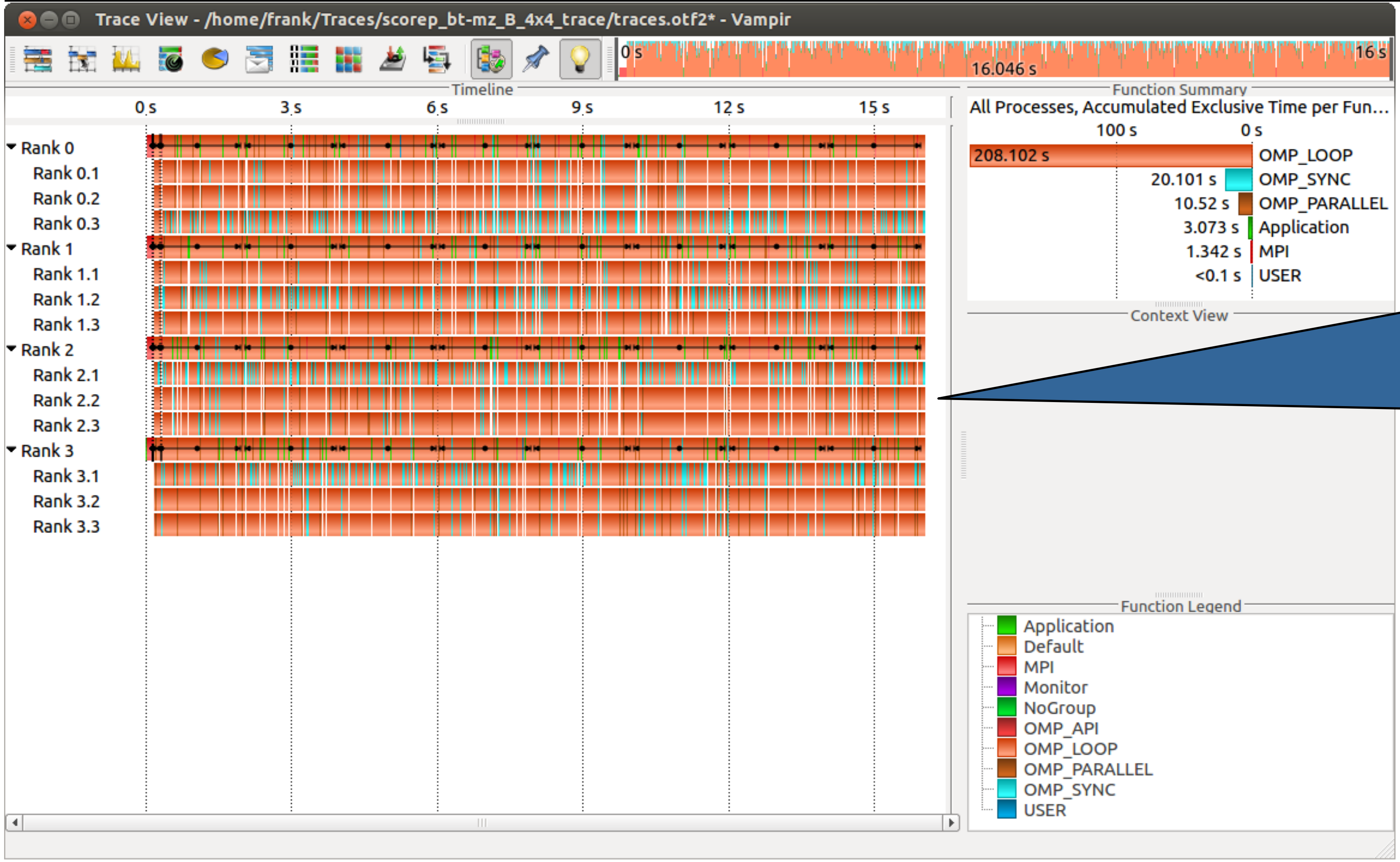

Vampir - [Trace View - /home/dolescha/tracefiles/feature-traces/wrf-p64-io-mem-rusage/wrf.1h.otf]
File View Help
View Chart Filter
Timeline
26.860 s 26.865 s 26.870 s 26.875 s 26.880 s 26.885 s 26.890 s
Process 0
Process 1
Process 2
Process 3
Process 4
Process 5
Process 6
Process 7
Process 8
Process 9
Process 10
Process 11
Process 12
Process 13
Process 14
Process 15
Process 16
Process 17
Process 18
Process 19
Process 20
Process 21
Process 22
Process 23
Process 24
MPI_Wait
solve_em_
module_em_mp_rk_step_prep_
Function Legend
Application
DYN
I/O
IO
MEM
MPI
PHYS
VT_API
WRF
Context View
Master Timeline
Property
Value
Display
Master Timeline
Type
Function
Function Name
MPI_Wait
Function Group
MPI
Interval Begin
26.872264 s
Interval End
26.885236 s
Duration
0.012972 s
Source File
Source Line


Timeline
34.149119 s +50 µs +100 µs +150 µs +200 µs +250 µs +300 µs
Process 0 MPI_Wait
Process 1 MPI_Isend MPI_Wait MPI_Wait
Process 2 MPI_Wait SOLVE_EM
Process 3 SOLVE_EM
Process 4 MPI_Wait
Process 5 SOLVE_EM MPI_Irecv MPI_Isend
Process 6 MPI_Irecv MPI_Isend
Process 7 MPI_Wait
Process 8 MPI_Wait SOLVE_EM
Process 9 MPI_Wait
Process 10 MPI_Irecv MPI_Isend
Process 11 MPI_Wait
Process 12 COUPLE_MOMENTUM
Process 13 SOLVE_EM
Process 14 SOLVE_EM
Process 15 SOLVE_EM
34.149209 s
Function Summary
All Processes, Accumulated Exclusive Time per Function
2 ms 1 ms 0 ms
2.37 ms MPI_Wait
1.708 ms SOLVE_EM
550.796 µs MPI_Isend
445.604 µs MPI_Irecv
340.607 µs COUPLE_...OMENTUM
25.423 µs CALCULATE_FULL
9.35 µs RK_STEP_PREP
Context View
Master Tim...
Property Value
Display Master Timeline
Type Message
Message Type Point to point
Origin Process 1
Destination Process 2
Communicator MPI Communicator 0
Tag 2
Start Time 34.149161 s
Arrival Time 34.149379 s
Duration 218.05 µs
Size 199.335938 KiB
Data Rate 892.749376 MiB/s

# Visualization of the NPB-MZ-MPI / BT trace

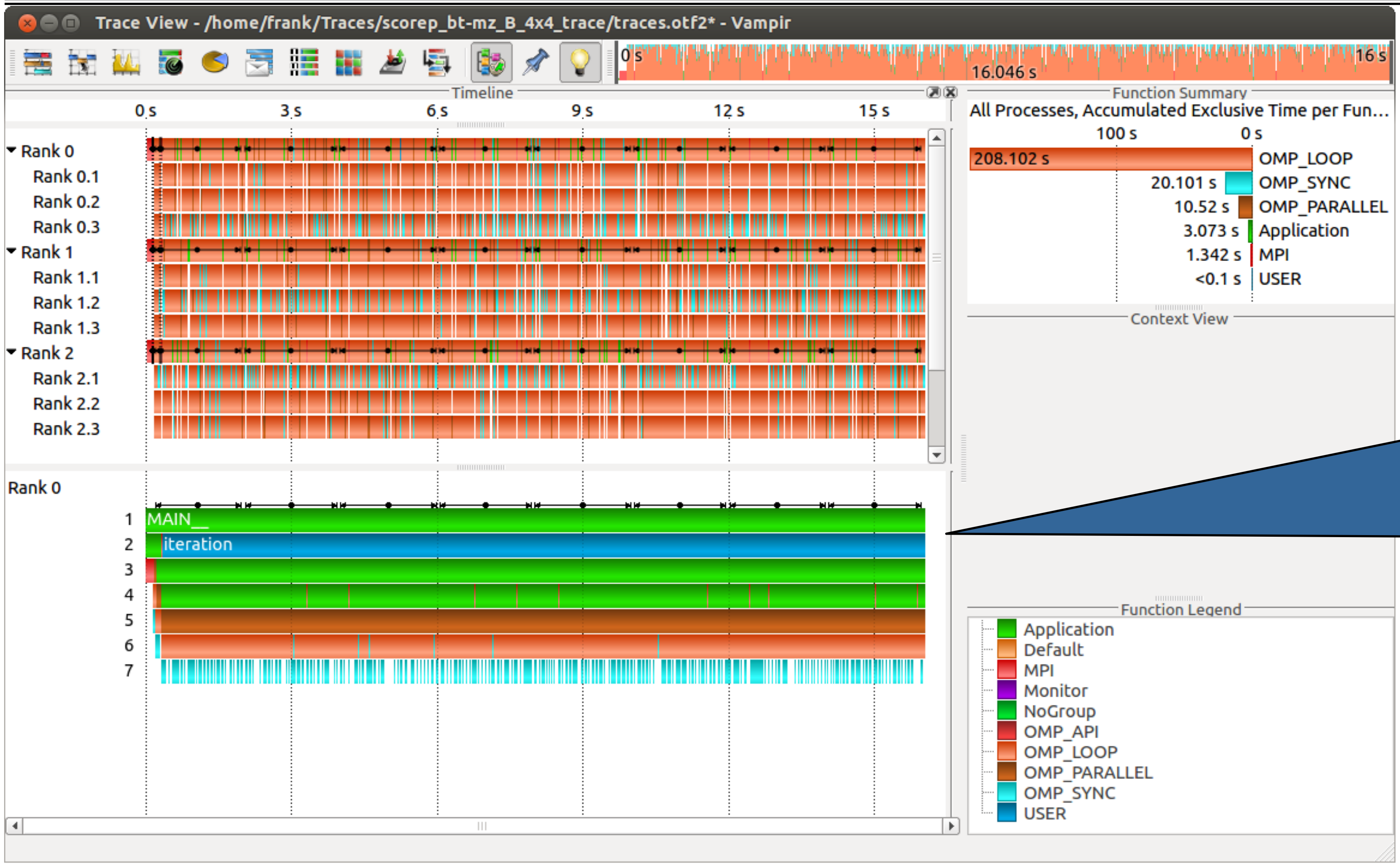


# Visualization of the NPB-MZ-MPI / BT trace
## Master Timeline




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

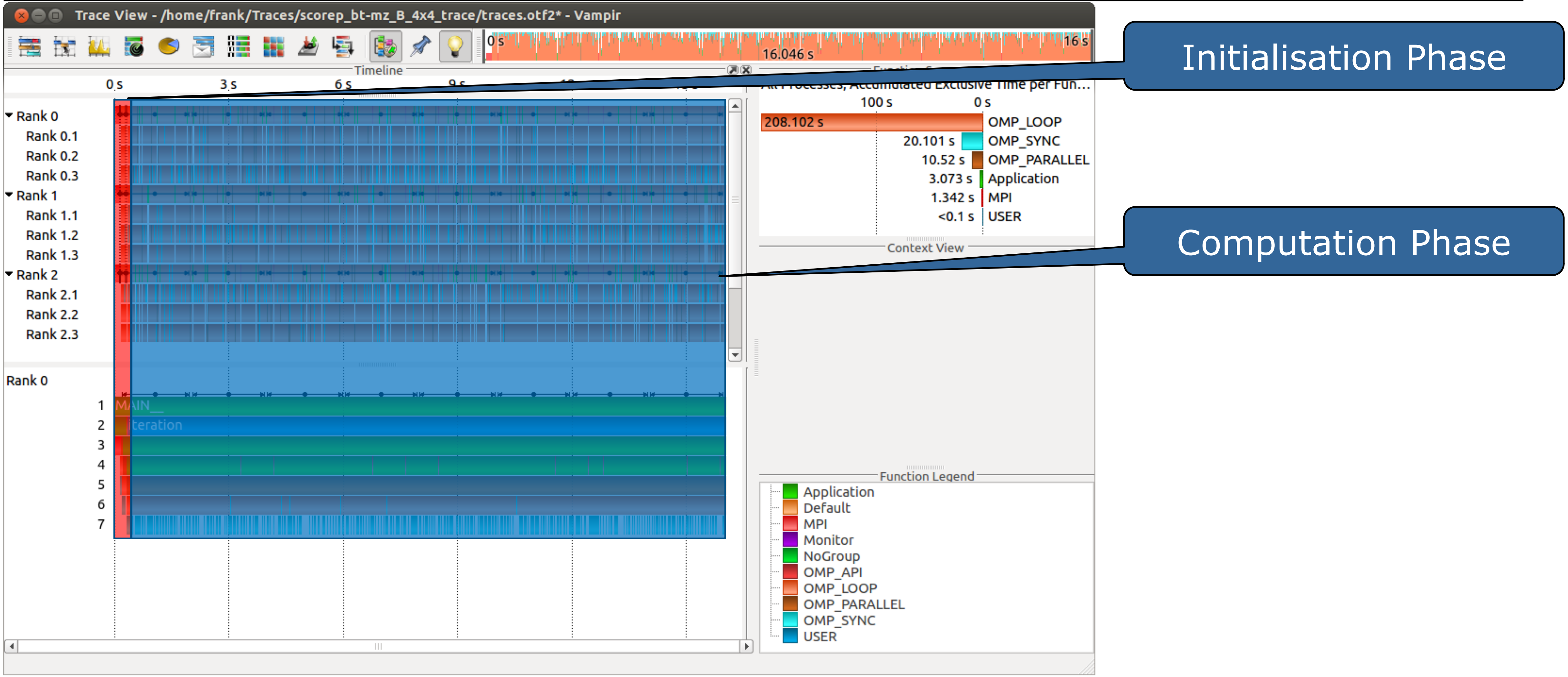
# Visualization of the NPB-MZ-MPI / BT trace
## Process Timeline



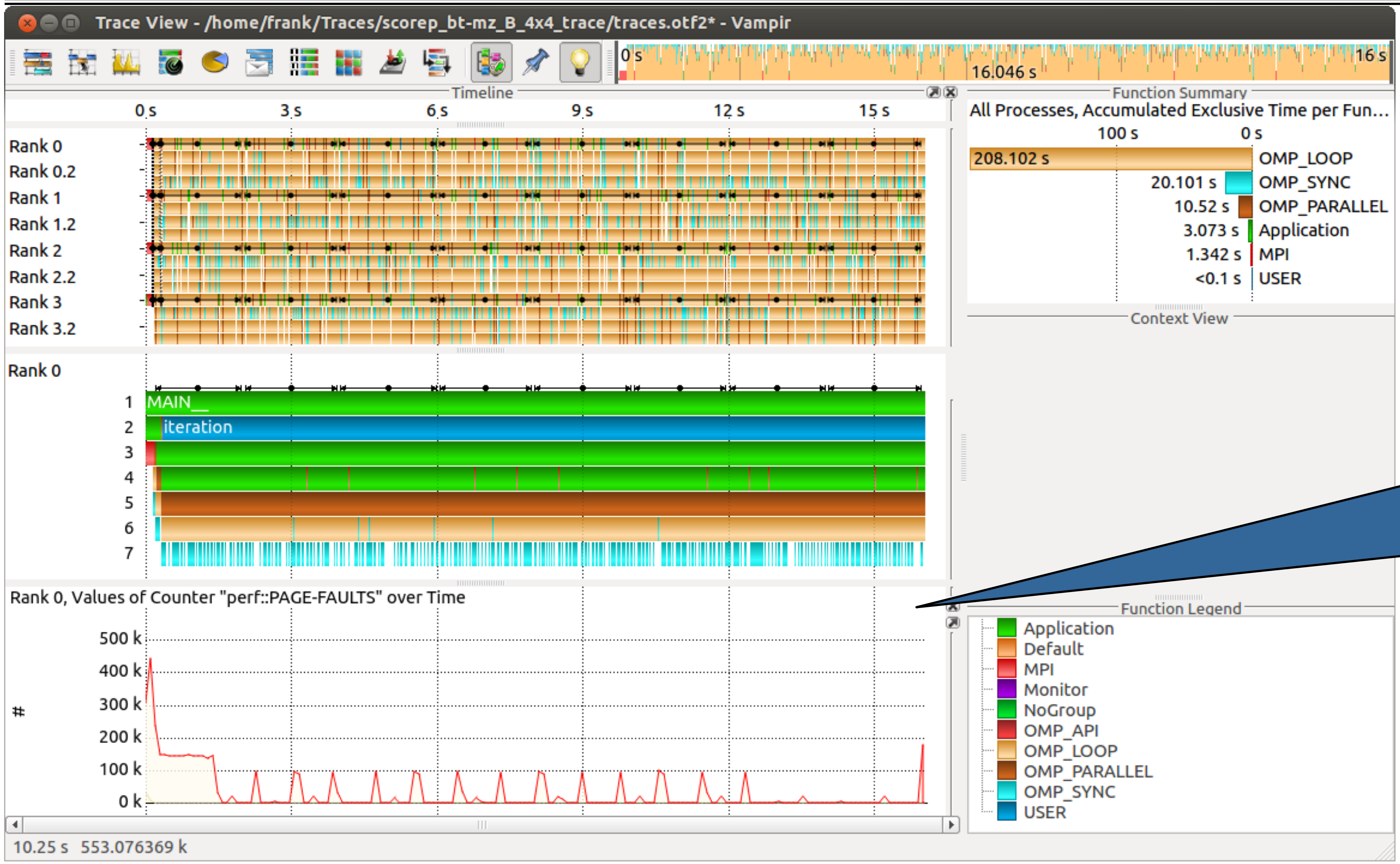

Detailed information about different levels of function calls in a stacked bar chart for an individual process.

# Visualization of the NPB-MZ-MPI / BT trace

## Typical program phases

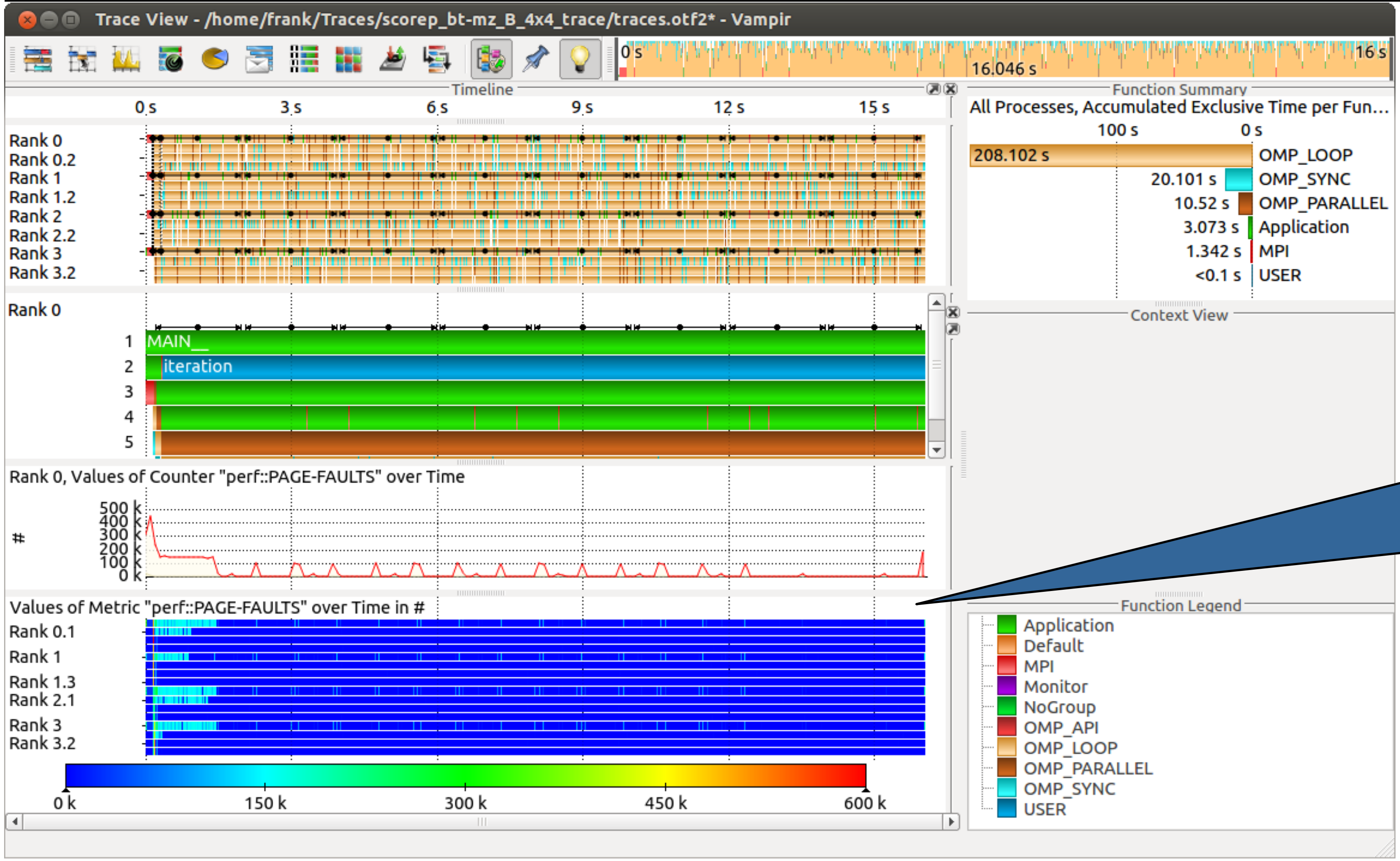


# Visualization of the NPB-MZ-MPI / BT trace
## Counter Data Timeline




Detailed counter information over time for an individual process.

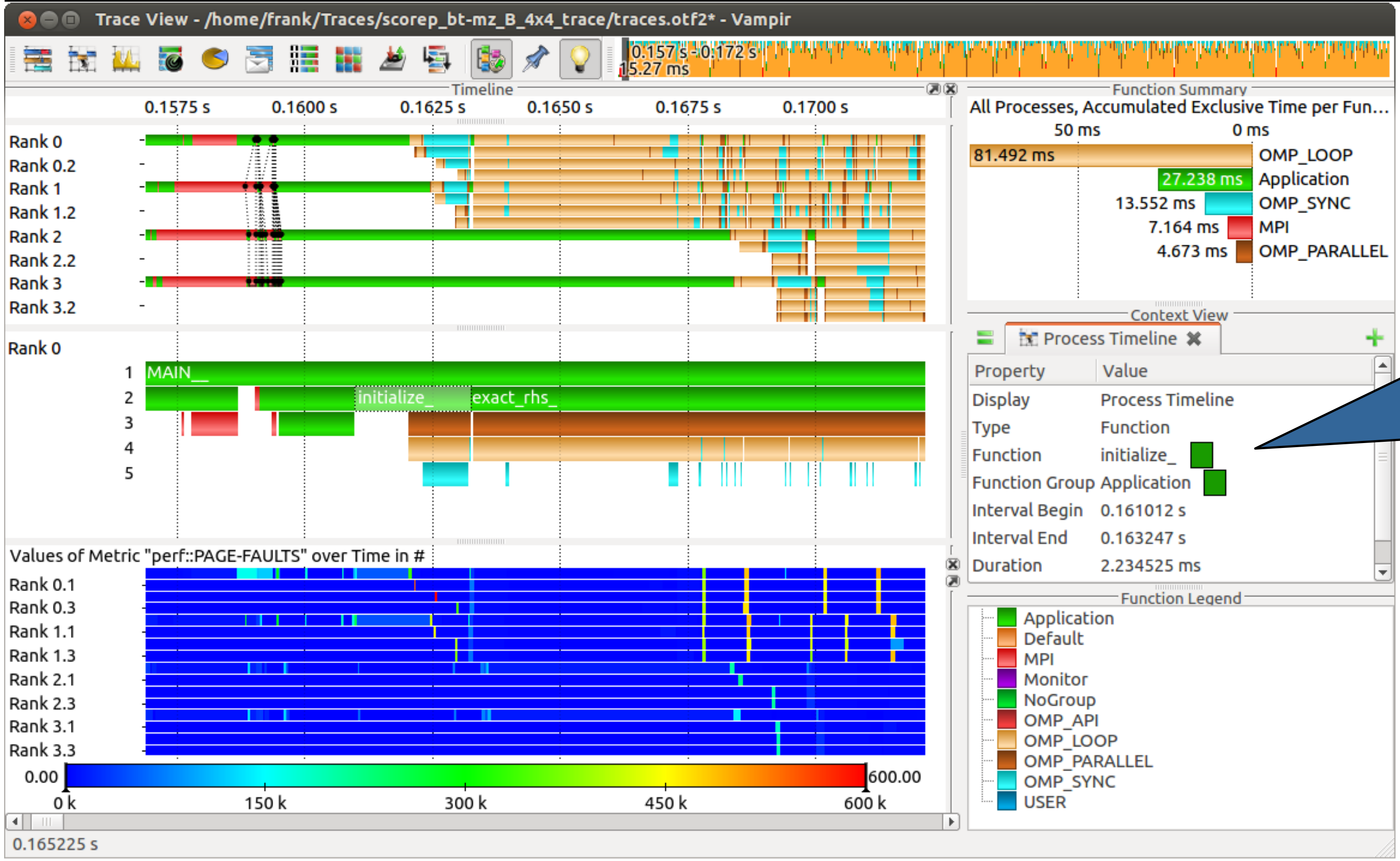
# Visualization of the NPB-MZ-MPI / BT trace
## Performance Radar




Detailed counter information over time for a collection of processes.

# Visualization of the NPB-MZ-MPI / BT trace

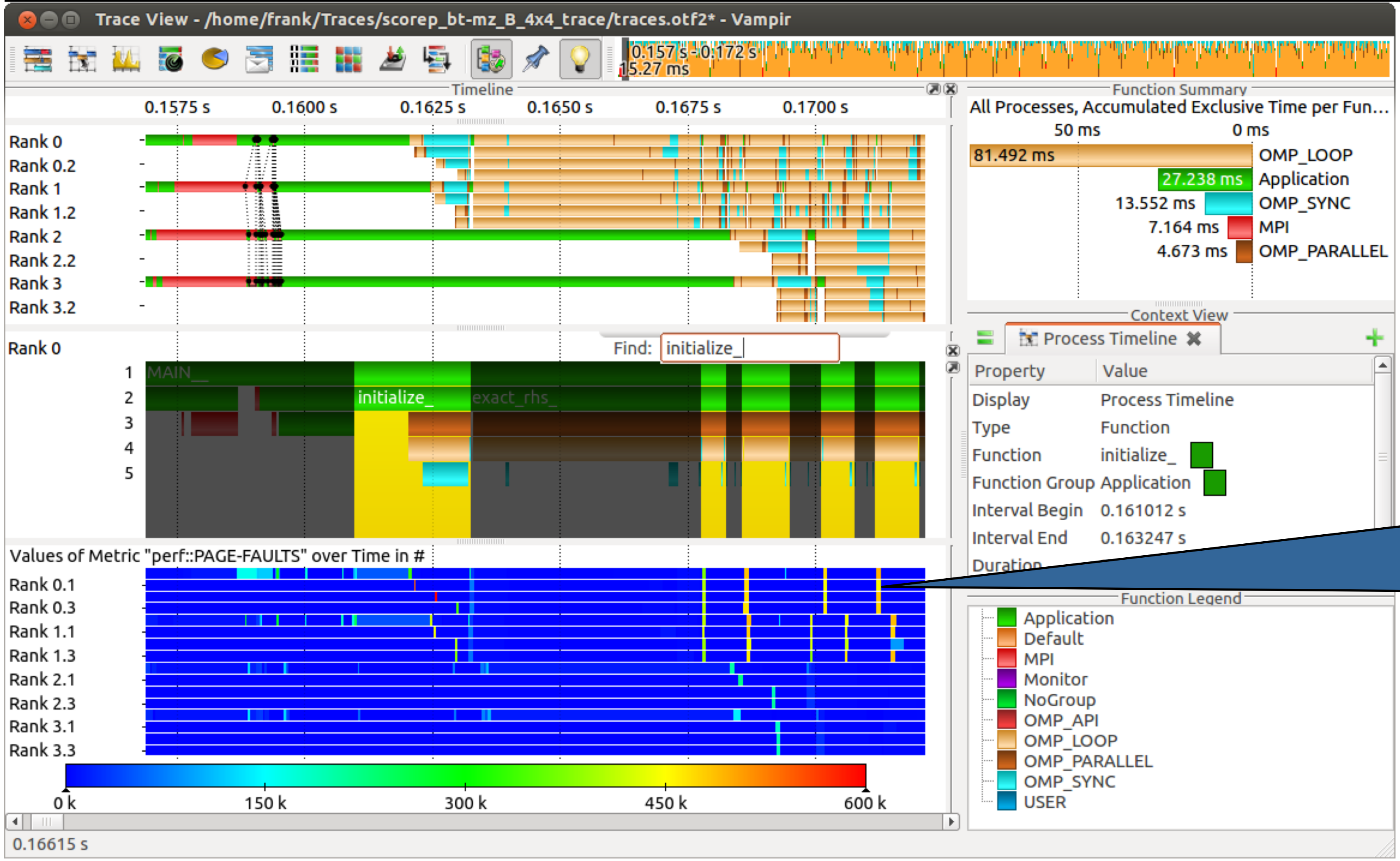
## Zoom in: Inititialisation Phase




Context View: Detailed information about function “initialize_”.

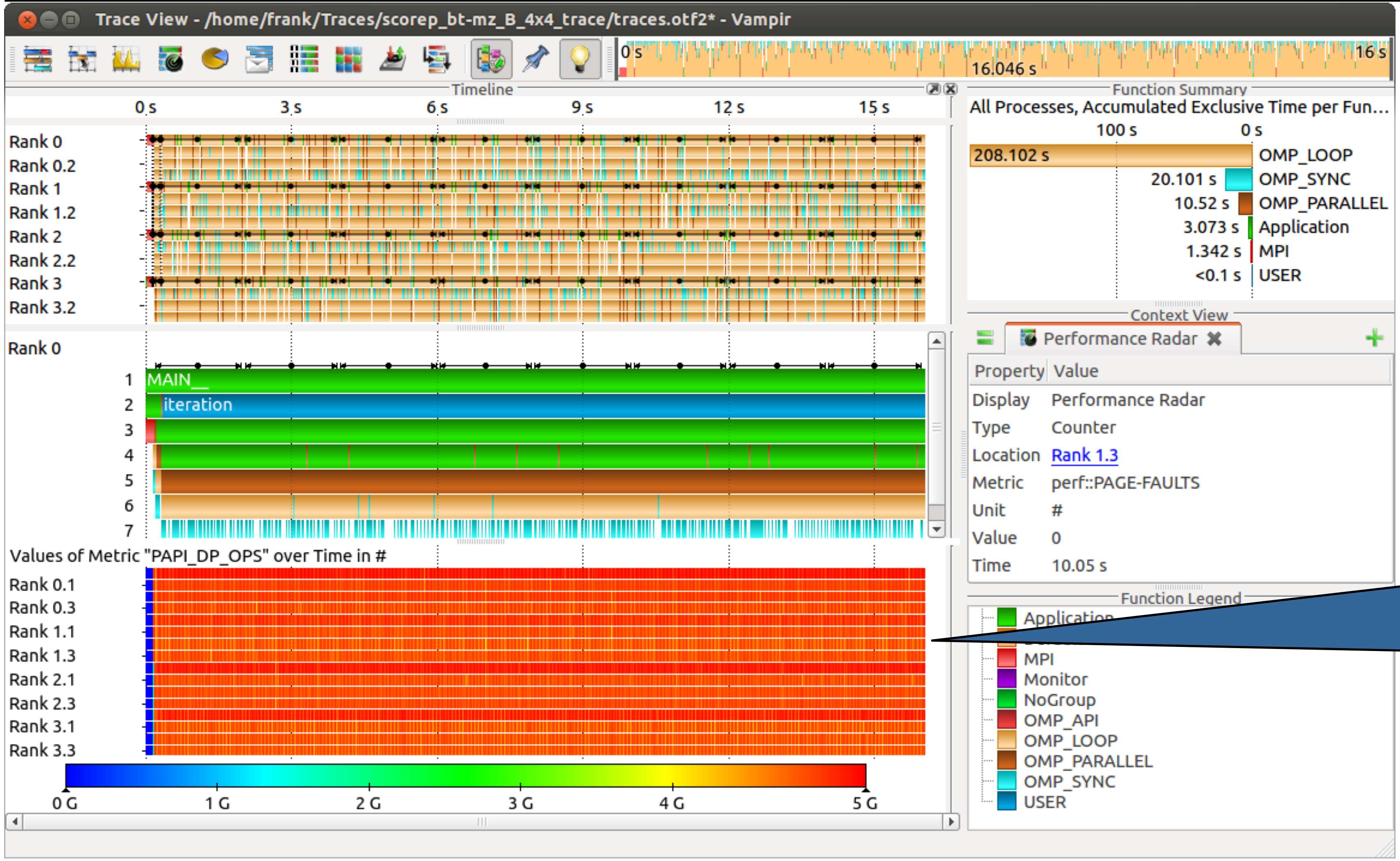
# Visualization of the NPB-MZ-MPI / BT trace

## Find Function




Execution of function “initialize_” results in higher page fault rates.

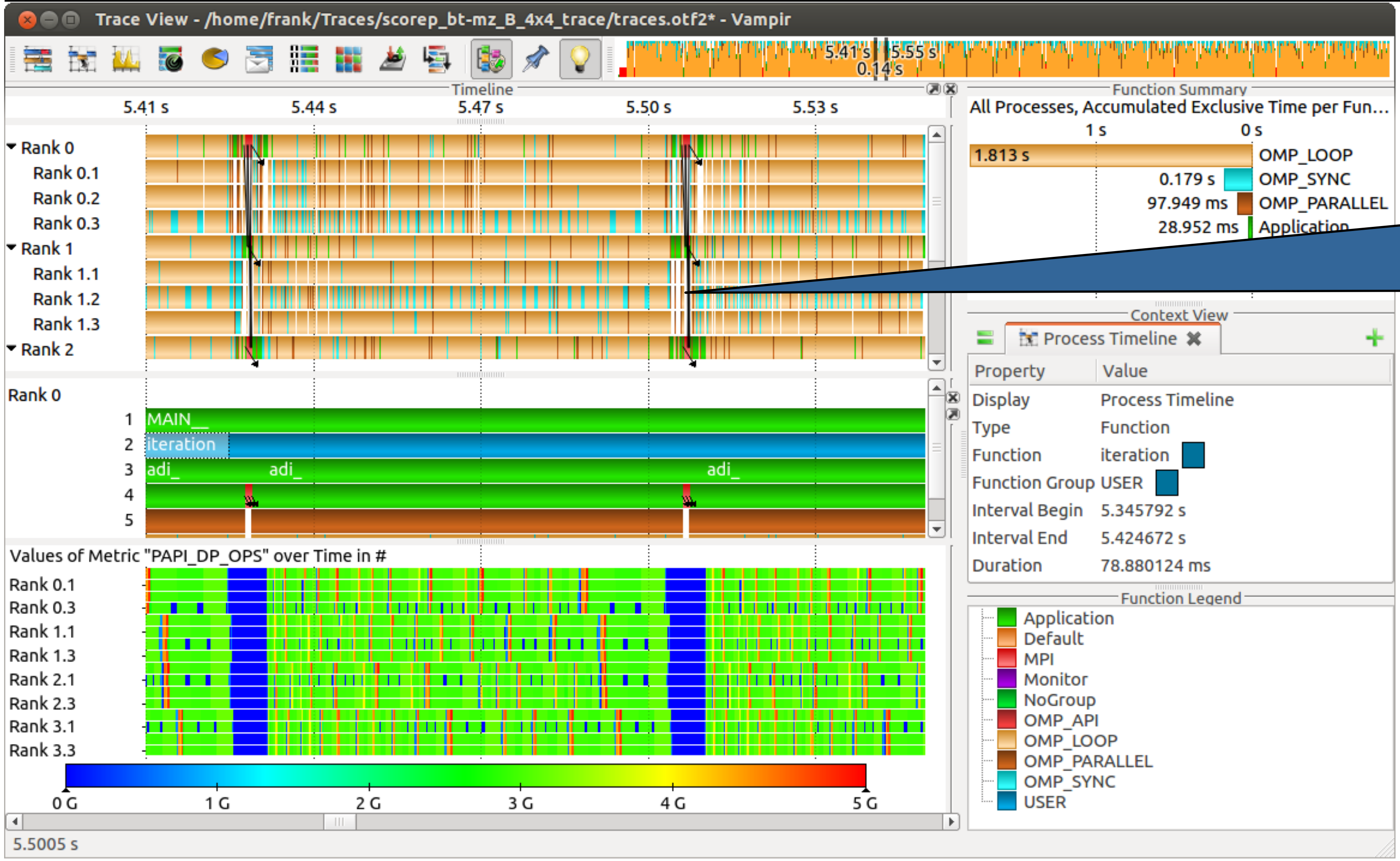
# Visualization of the NPB-MZ-MPI / BT trace
## Computation Phase




Computation phase results in higher floating point operations.

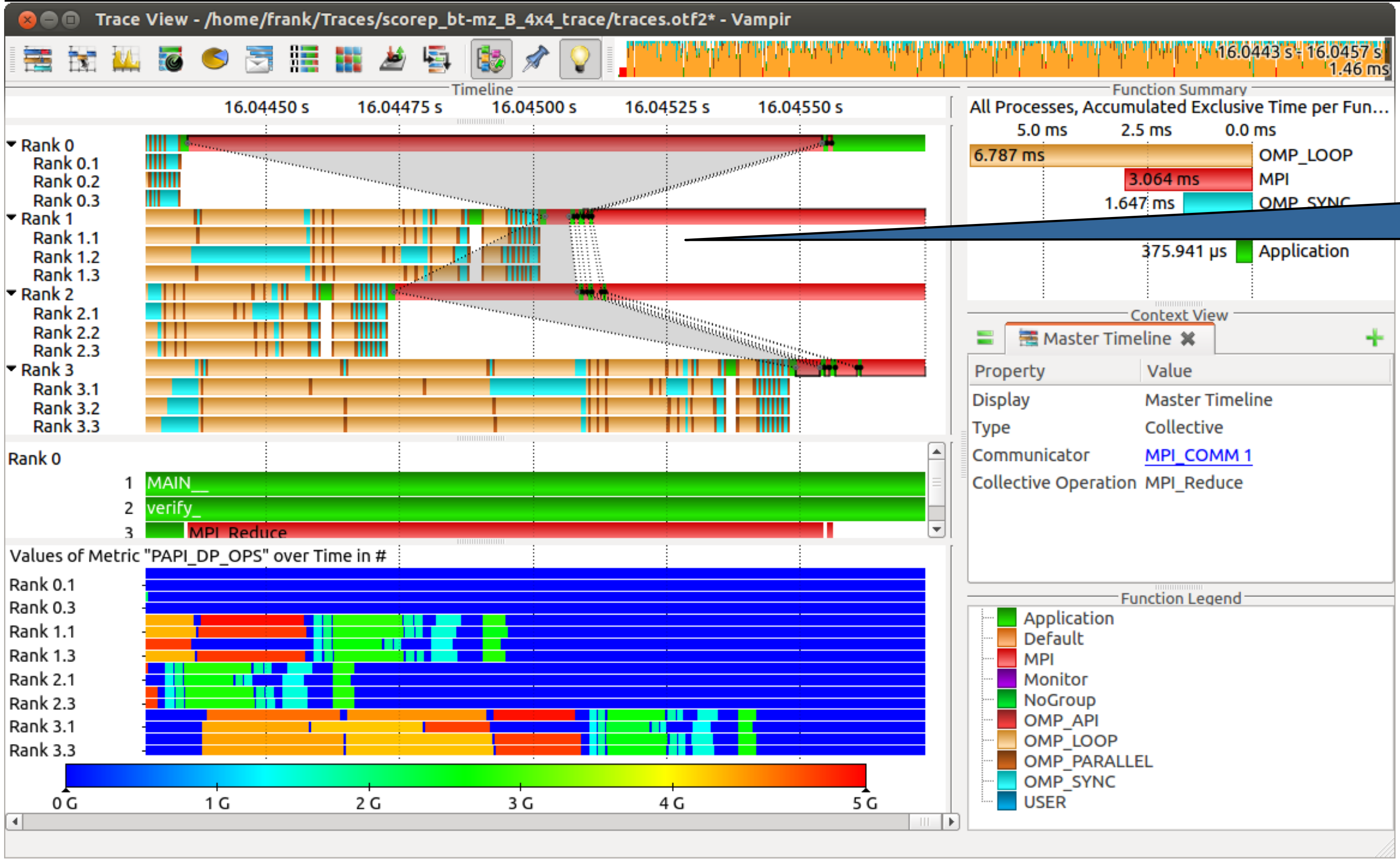
# Visualization of the NPB-MZ-MPI / BT trace

## Zoom in: Computation Phase



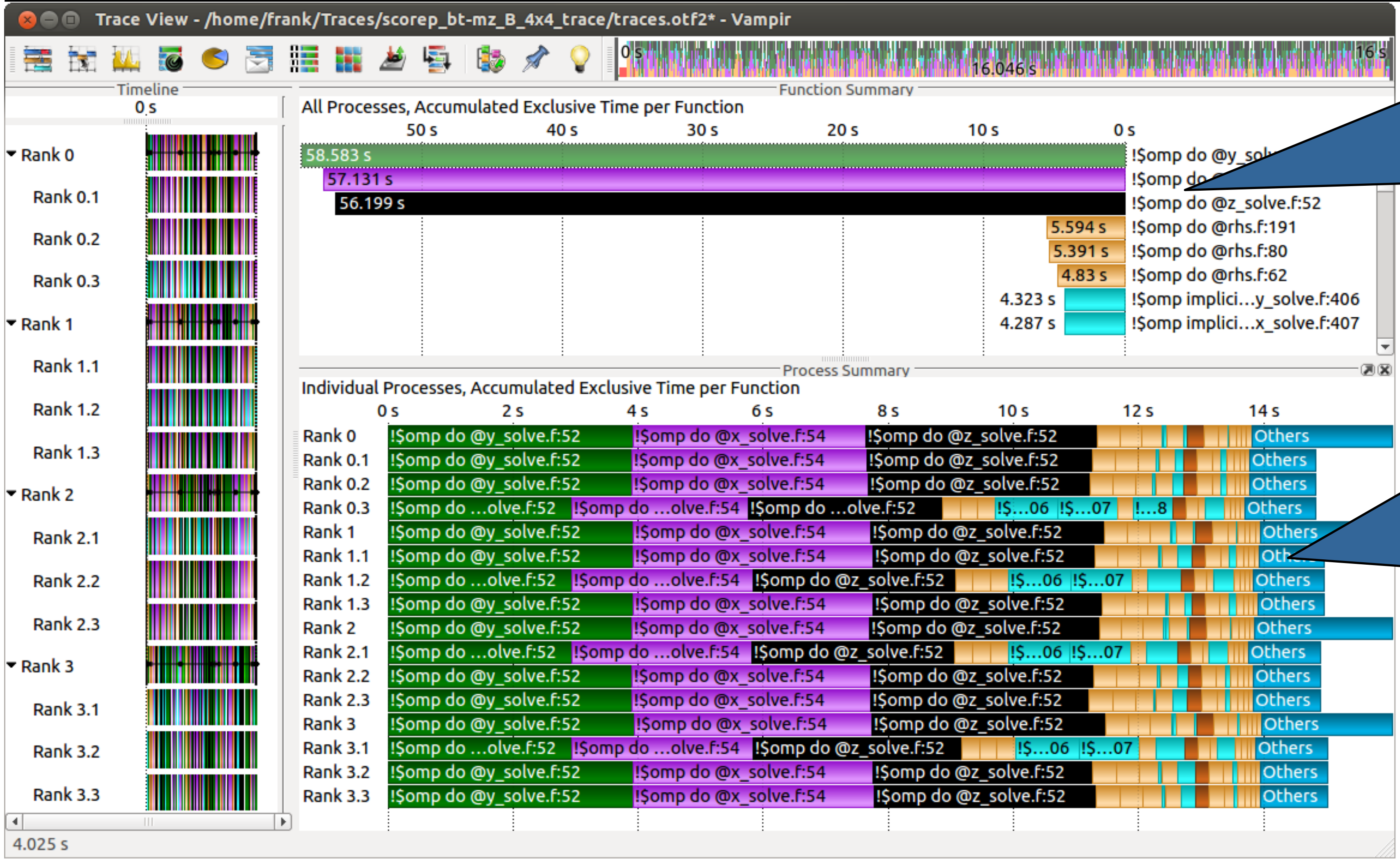

MPI communication results in lower floating point operations.

# Visualization of the NPB-MZ-MPI / BT trace
## Zoom in: Finalisation Phase




"Early reduce" bottleneck.

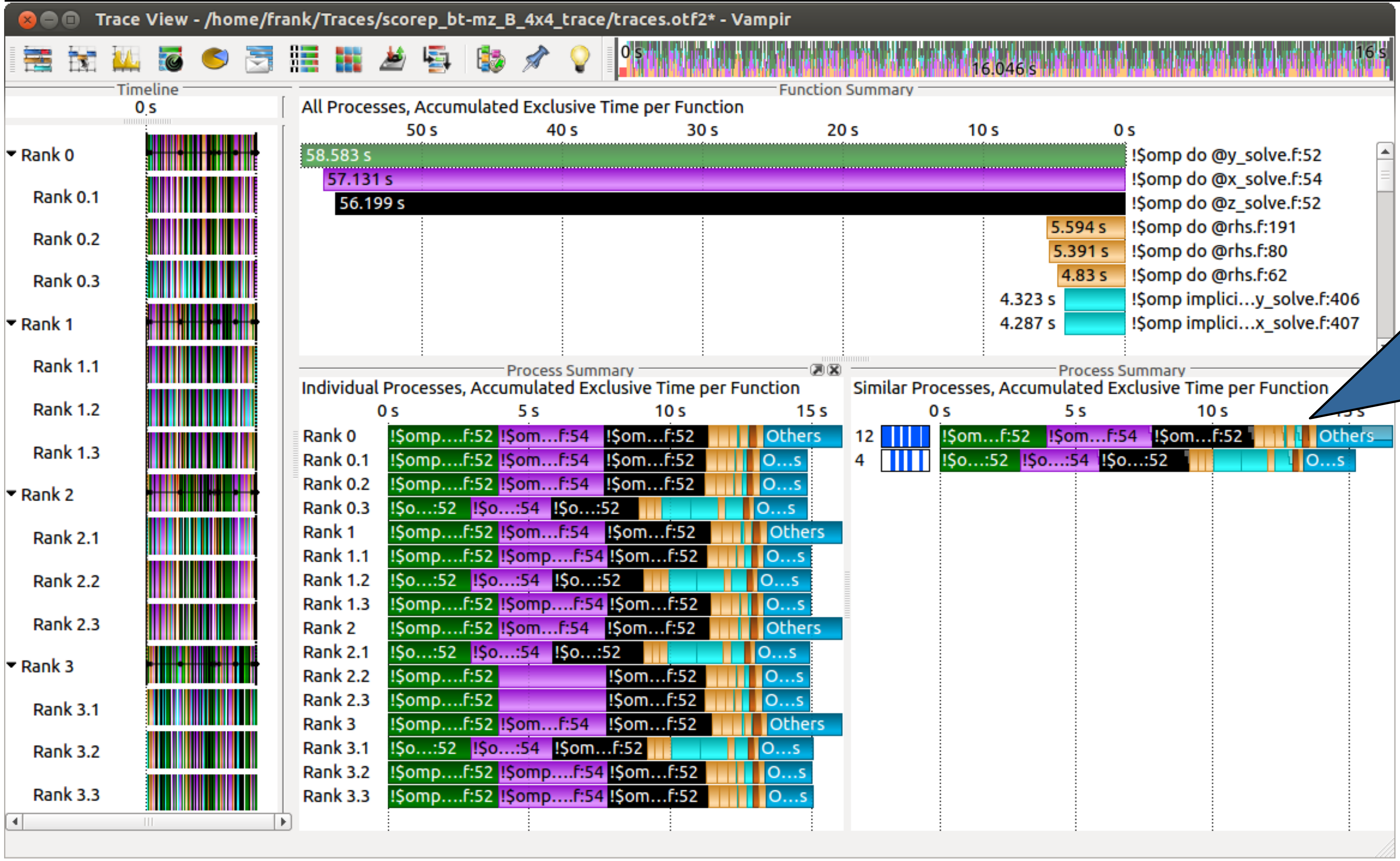
# Visualization of the NPB-MZ-MPI / BT trace
## Process Summary




**Function Summary:** Overview of the accumulated information across all functions and for a collection of processes.

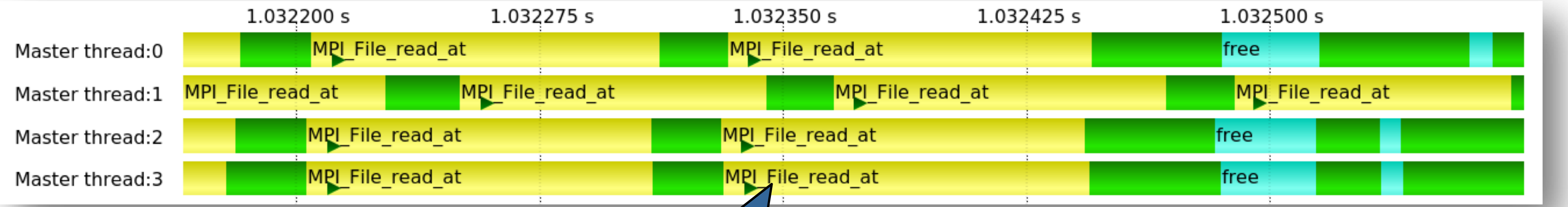
**Process Summary:** Overview of the accumulated information across all functions and for every process independently.

# Visualization of the NPB-MZ-MPI / BT trace
## Process Summary

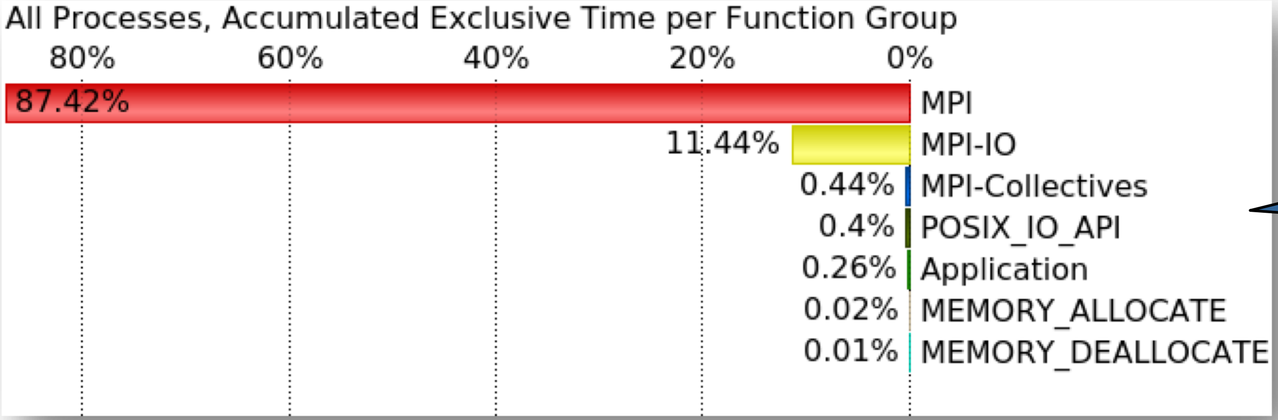



Find groups of similar processes and threads by using summarized function information.

# I/O Features
## Operations over Time

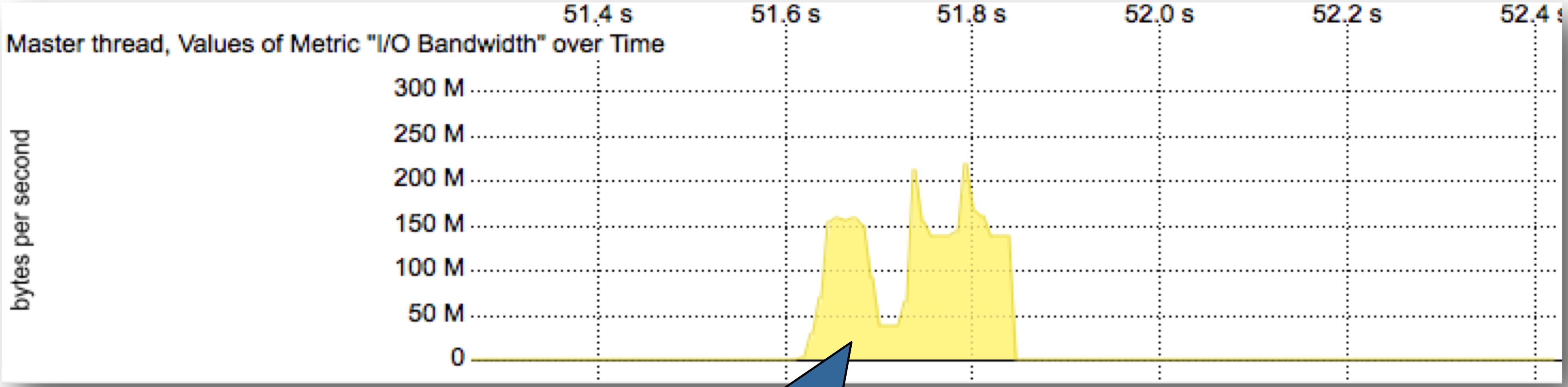



Individual I/O operation




I/O runtime contribution

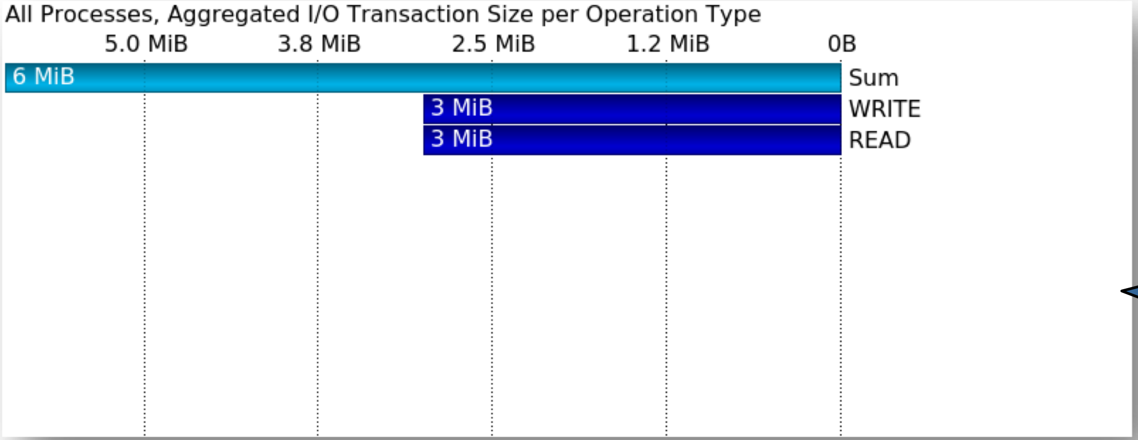
# I/O Features
## Data Rates over Time



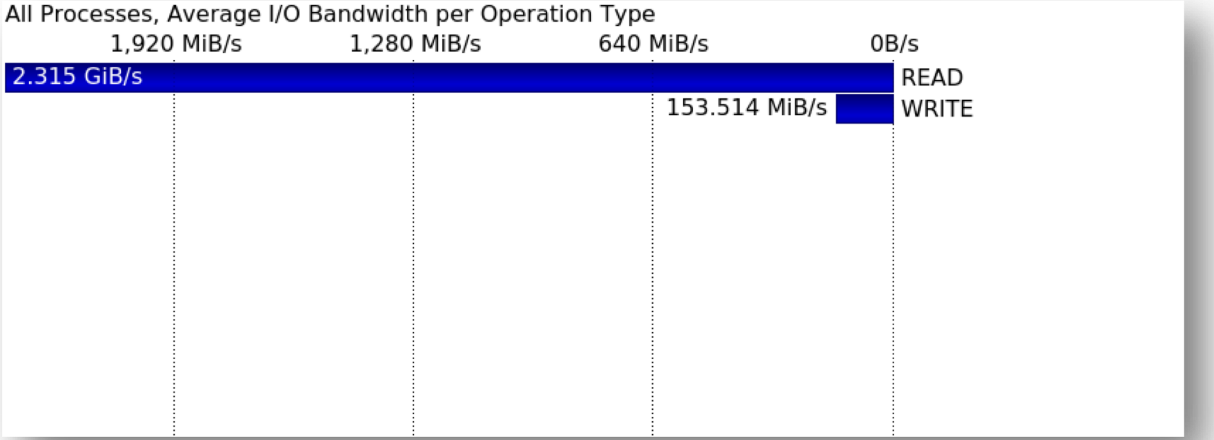

I/O data rate of single thread

# I/O Features

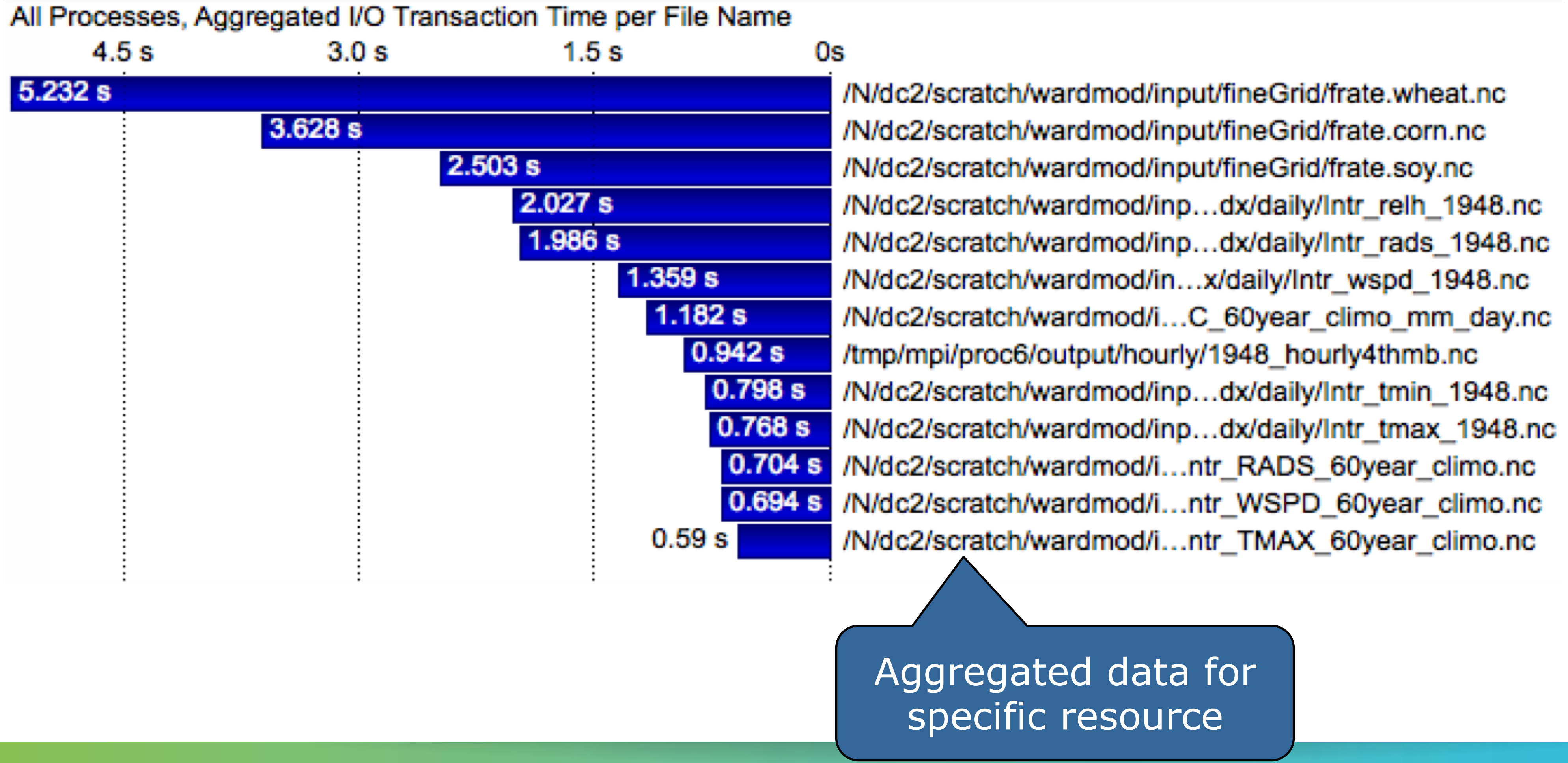
## Summaries with Totals



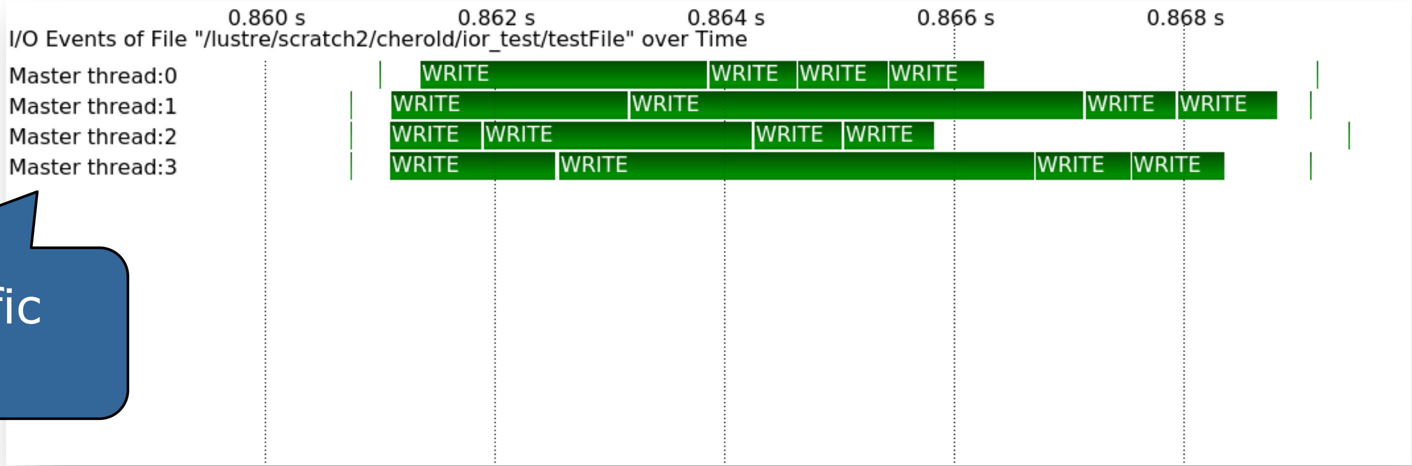

Other metrics:
IOPS
I/O time
I/O size
I/O bandwidth



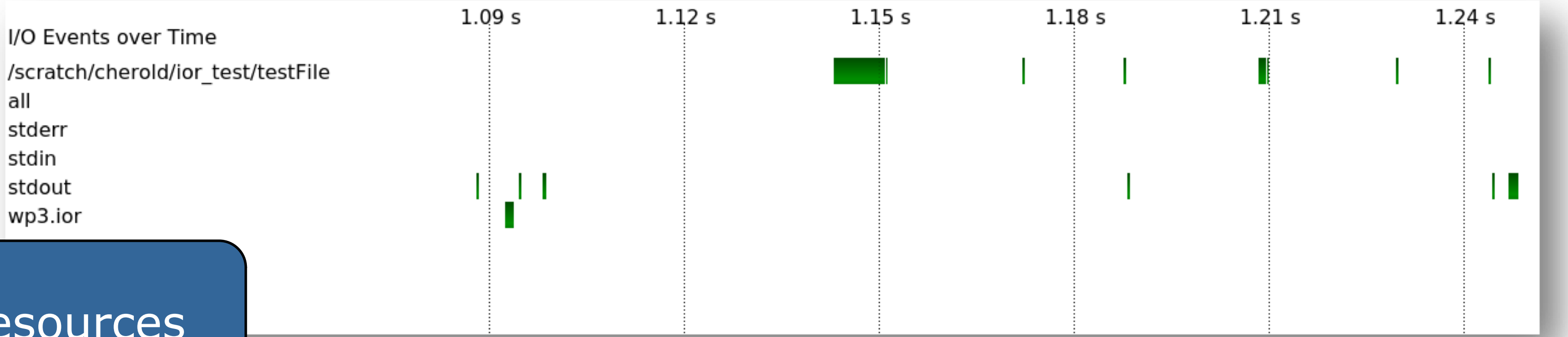
# I/O Features
## Summaries per File



# I/O Features
## Operations per File




Focus on specific resource




Show all resources

# Summary and Conclusion

# Summary

- Vampir & VampirServer
  - Interactive trace visualization and analysis
  - Intuitive browsing and zooming
  - Scalable to large trace data sizes (20 TiByte)
  - Scalable to high parallelism (200,000 processes)
- Vampir for Linux, Windows, and Mac OS X

SPONSORED BY THE

Federal Ministry
of Education
and Research

INDIANA UNIVERSITY

SILC

OAK RIDGE
National Laboratory

LMAC

READEX
Runtime Exploitation of Application Dynamism
for Energy-efficient eXascale computing

eeClust
Energy-Efficient Cluster Computing

nextgenio

Score-E

VAMPIR

European Commission

Horizon 2020
European Union funding
for Research & Innovation

Microsoft

ITEA 2

SEVENTH FRAMEWORK PROGRAMME

H4H

ParMA

http://www.vampir.eu

vampirsupport@zih.tu-dresden.de