TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing (ZIH)

Performance and Correctness with the Vampir Tool-Suite and MUST

Dr. Matthias S. Müller
Tobias Hilbrich

Email: {matthias.mueller, tobias.hilbrich}@tu-dresden.de

ZIH
Center for Information Services & High Performance Computing

Content

- Welcome & Motivation
- Performance Optimization with Vampir
- Runtime Error Detection with MUST
- Conclusions & Wrap-Up

Part I: Welcome & Motivation

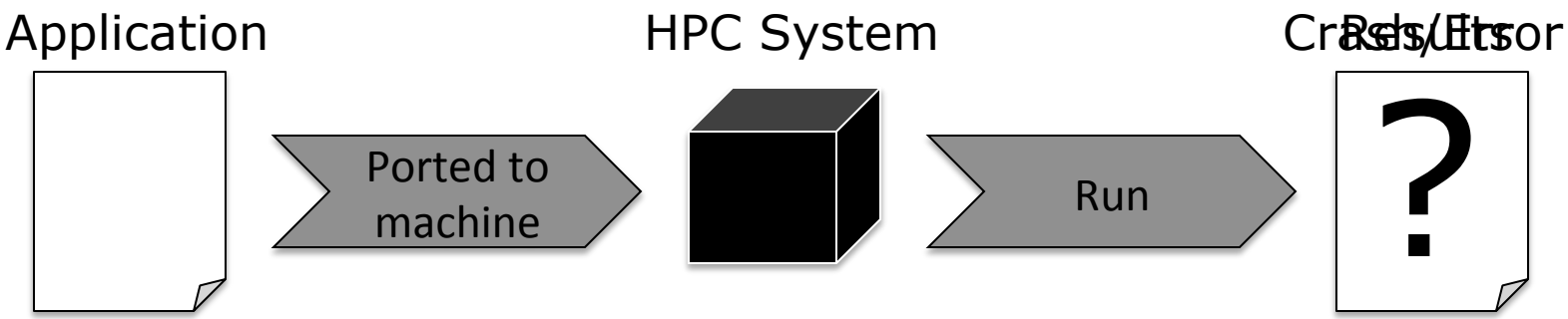
Dr. Matthias S. Müller
Tobias Hilbrich

Email: {matthias.mueller, tobias.hilbrich}@tu-dresden.de

ZIH
Center for Information Services & High Performance Computing

Motivation

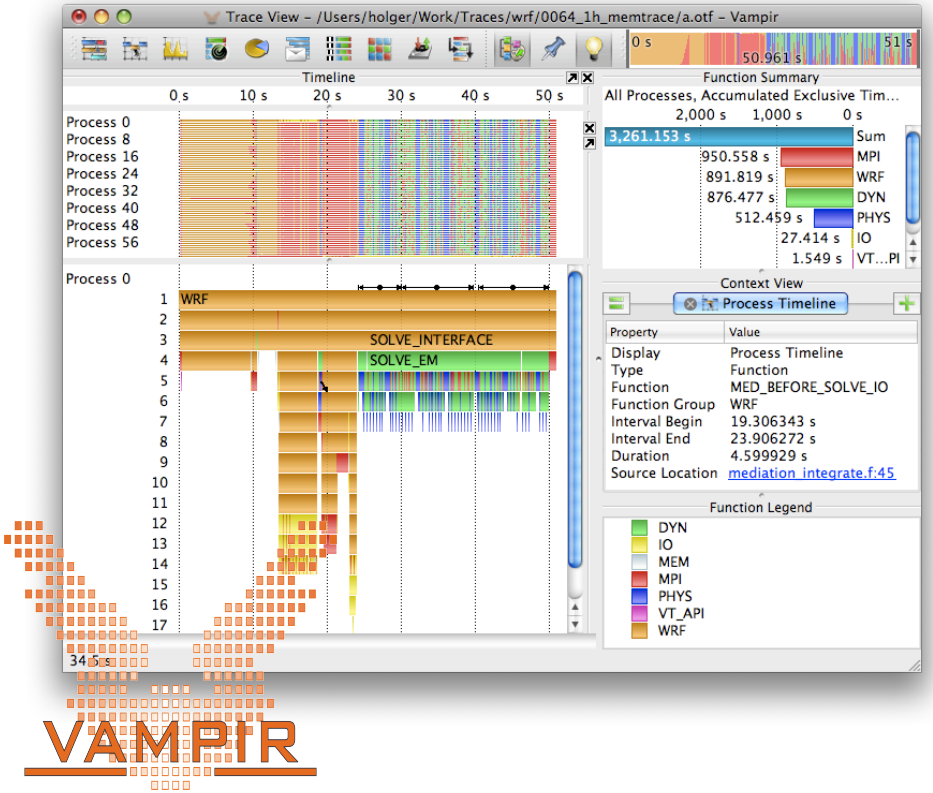
- High performance computing without tools:

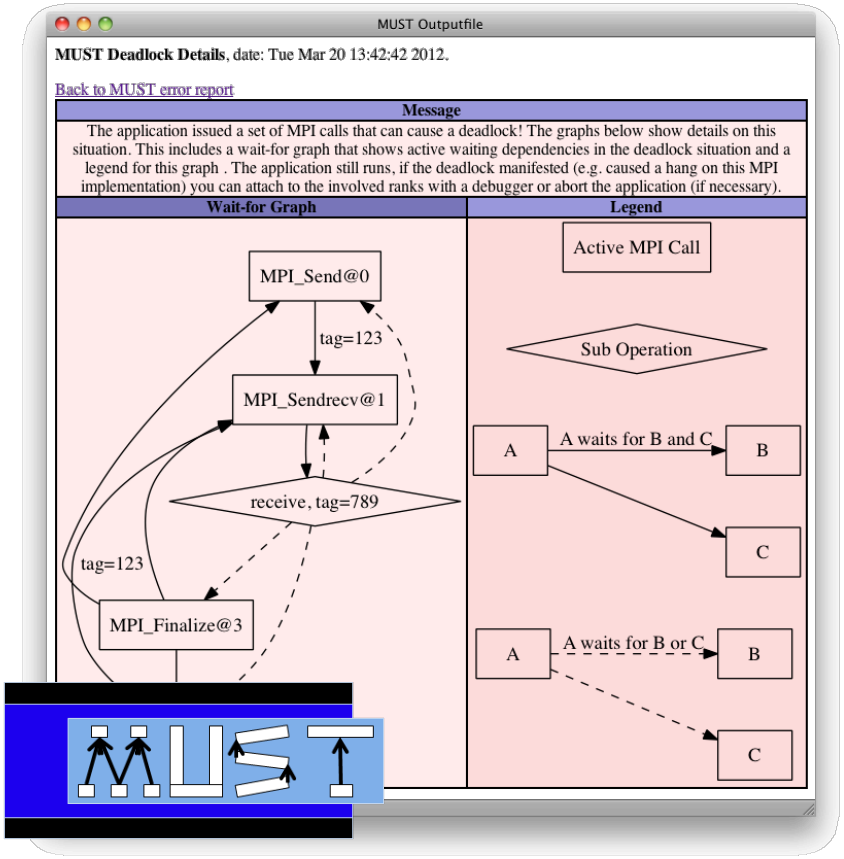


- Questions:
  - Did I utilize the machine efficiently?
  - Are my results correct?
  - Why aren't I getting a result?
- Tools help developers understand HPC systems

Tools!

- We got tools to assist you:



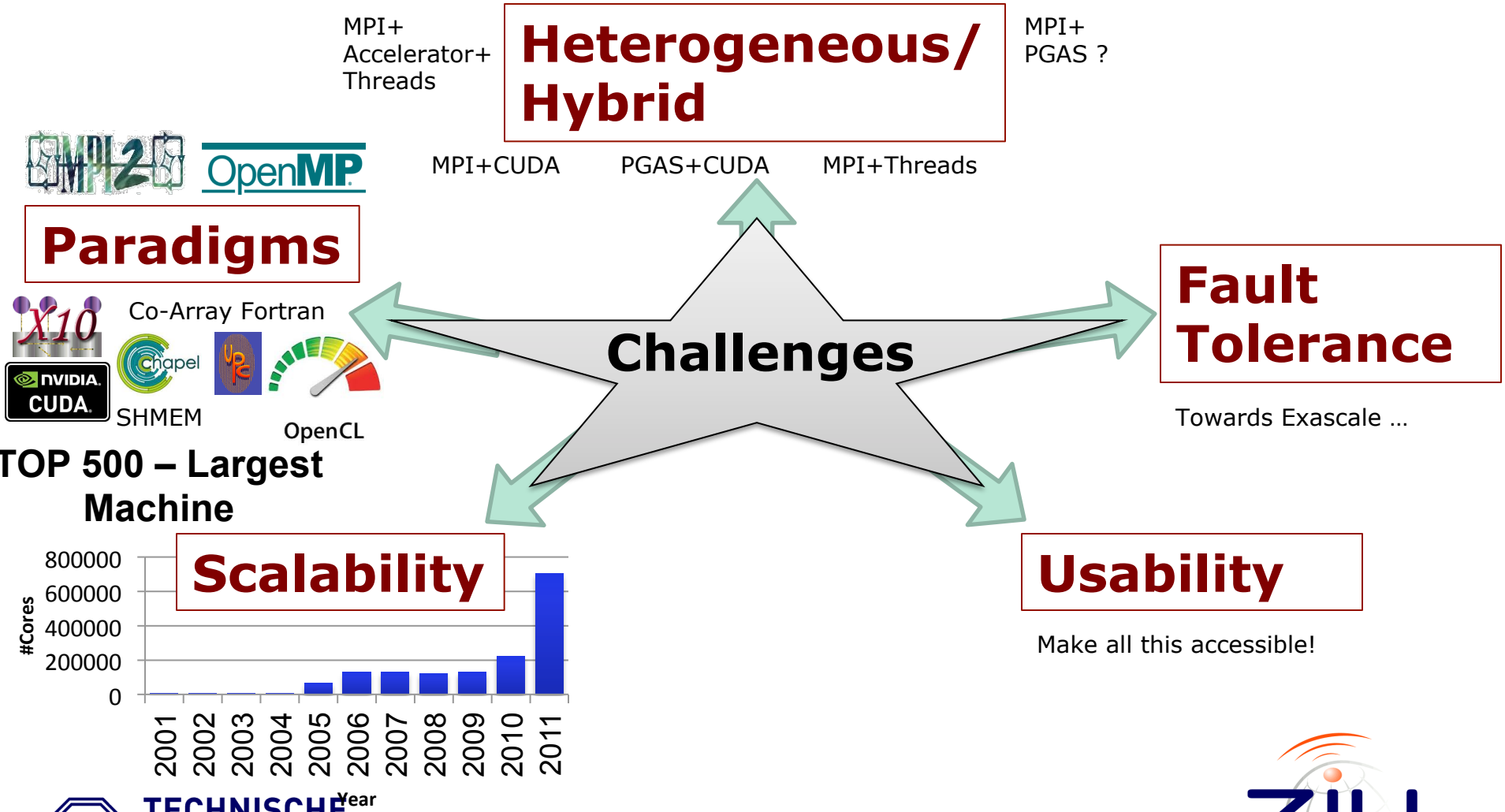


Vampir: Performance optimization

MUST: Correct MPI usage

Challenges

- HPC systems evolve; Tools need to adapt:



Agenda

- We want to cover:
  - How can our tools help you
  - Basic tool usage
  - New features and success stories
  - Roadmaps
  - Detailed training in individual sessions

Part II: Performance Optimization with Vampir

Dr. Matthias S. Müller
Tobias Hilbrich

Email: {matthias.mueller, tobias.hilbrich}@tu-dresden.de

ZIH
Center for Information Services & High Performance Computing

Content

- Overview
- Technology
- Example
- Tracing
- Visualization
- Meeting the Challenges
- Score-P
- Roadmap

Content

- Overview
- Technology
- Example
- Tracing
- Visualization
- Meeting the Challenges
- Score-P
- Roadmap

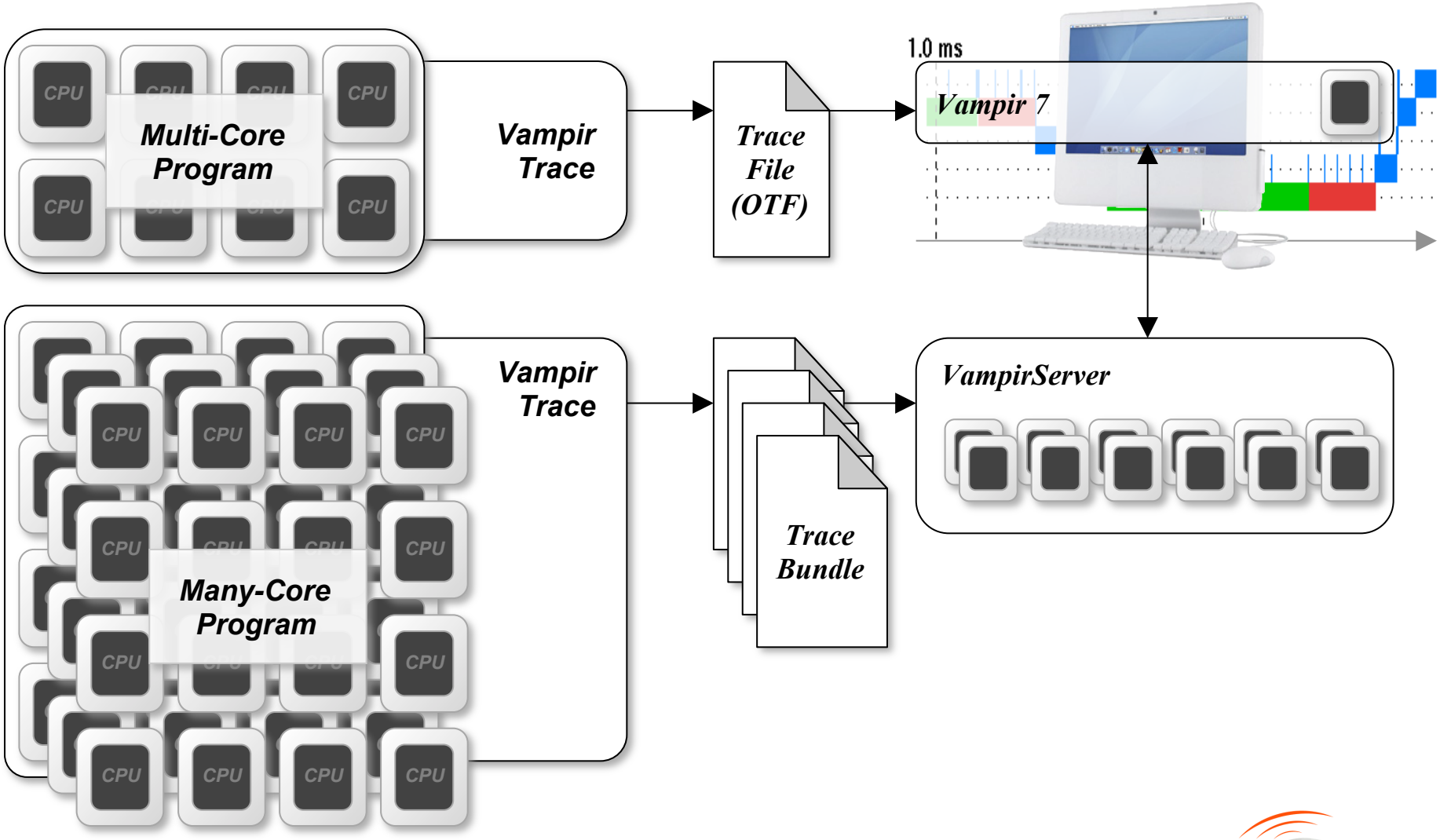
Overview

- Vampir: tool to optimize application performance
- Approach: Event tracing
- Insights with Vampir include:
  - Communication/Computation ratio
  - Communication statistics (Bandwith/Latency)
  - Inefficient communication patterns
  - Excessive synchronization time
  - Inefficient IO, or GPGPU usage
- However: Vampir is no automatic tool
  - A GUI guides the user to performance issues

Content

- Overview
- Technology
- Example
- Tracing
- Visualization
- Meeting the Challenges
- Score-P
- Roadmap

Technology

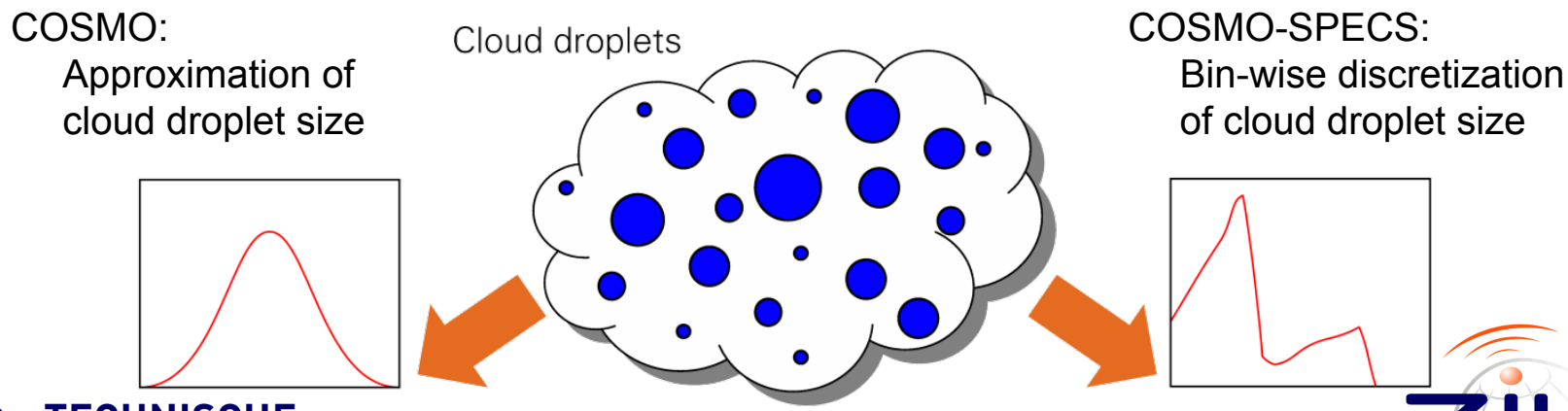

CPU
Multi-Core Program
Vampir Trace
Trace File (OTF)
1.0 ms
Vampir 7
Many-Core Program
Vampir Trace
Trace Bundle
VampirServer

Content

- Overview
- Technology
- Example
- Tracing
- Visualization
- Meeting the Challenges
- Score-P
- Roadmap

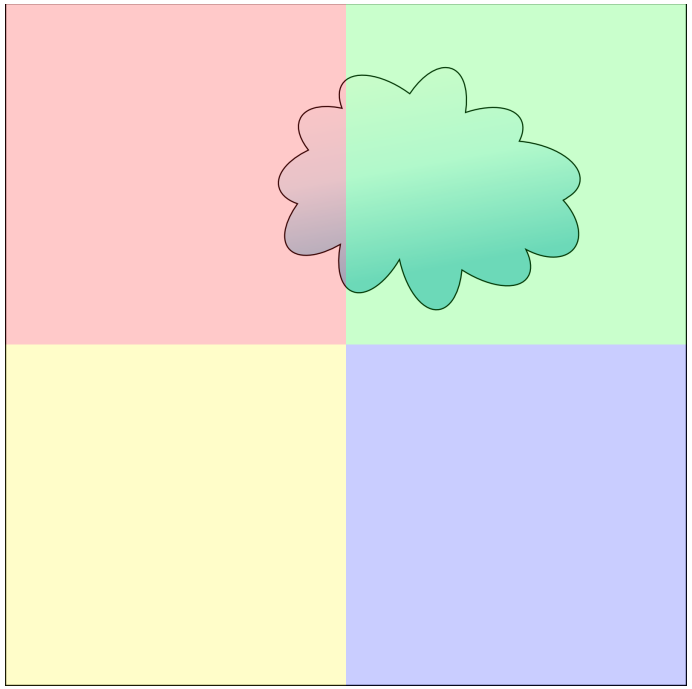
Example – COSMO-SPECS

- COSMO-SPECS is a coupled model system
  - COSMO: regional weather forecast model developed at the German Weather Service (DWD), formerly known as “Lokal-Modell” (LM)
  - SPECS: detailed cloud microphysics model developed at the Leibniz Institute for Tropospheric Research, Leipzig (IfT)
- Used for detailed study of the interactions between aerosols, clouds, and precipitation



Example – COSMO-SPECS Domain Decomposition

- COSMO computational grid:
  - 3D regular rectangular
- COSMO domain decomposition:
  - 2D (horizontal) decompostion into MxN processor domains
  - No dynamic load balancing
- SPECS uses the same data structures and decomposition
  - Technically “just” a replacement of COSMOs original cloud model
  - But causes **huge load imbalance**, since (heterogeneous) cloud cover determines local work load



Example – COSMO-SPECS Trace

- SGI Altix 4700, 100 cores
- Forecast time: 30min
- Run time: ~2:55h
- Usage of VT API to switch tracing off. Only the
  - first 3 time steps,
  - one time step at 25%, 50%, and 75% of the time steps, and
  - the last 3 time steps are recorded.
- Groups
  - METEO: COSMO weather model
  - MP: SPECS cloud microphysics calculation
  - MP_UTIL: Utilities for cloud microphysics (advection, clipping)
  - COUPLE: Adjustment of flux fields of COSMO (for microphysics)

Example – COSMO-SPECS Trace Visualization

First 3 time steps of COSMO-SPECS run

Last 3 time steps of COSMO-SPECS run

Heavy load imbalance

Cloud grows in grid cells of these MPI ranks

SPECS performs 20 sub-time steps per COSMO step

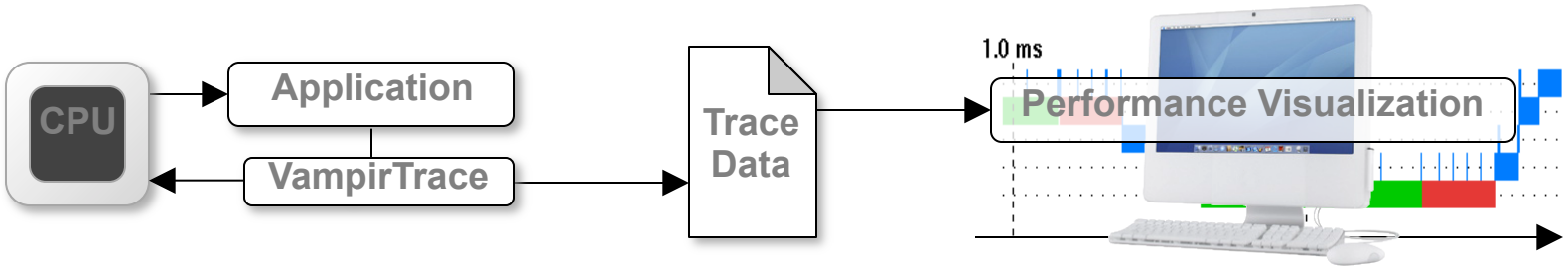
Everthing ok

56% MPI!

Content

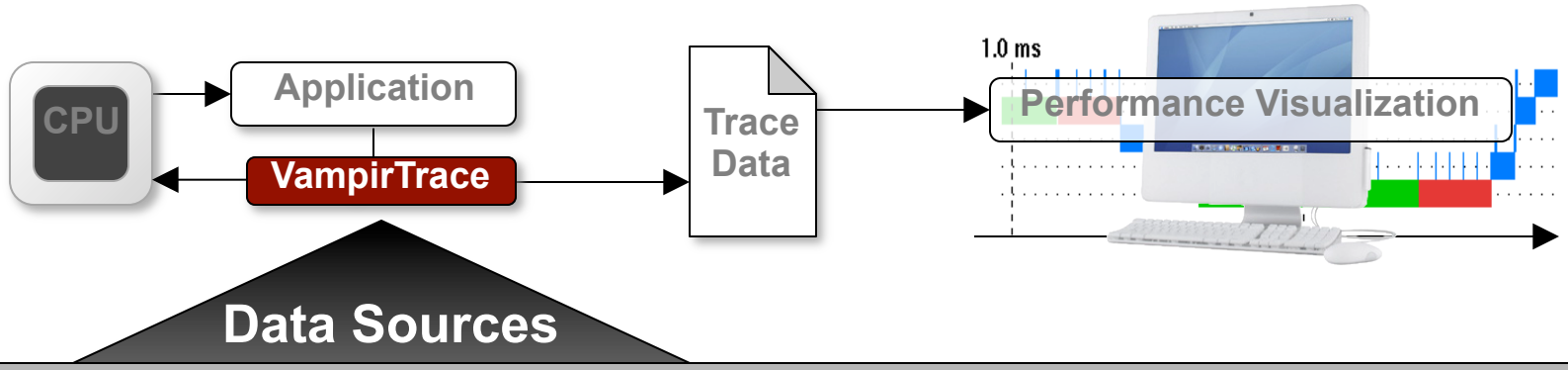
- Overview
- Technology
- Example
- Tracing
- Visualization
- Meeting the Challenges
- Score-P
- Roadmap

Tracing – Overview



- VampirTrace is a monitor for event tracing
- VampirTrace:
  - Open source, BSD license
  - Part of every OpenMPI distribution
- Usage:
  - Attach VampirTrace library to application
  - Various control and tuning options

Tracing – Instrumentation



Data Sources

Source level
- Compiler
- Manual
- OpenMP with Opari
- Tau PDT

Binary level
- Dyninst

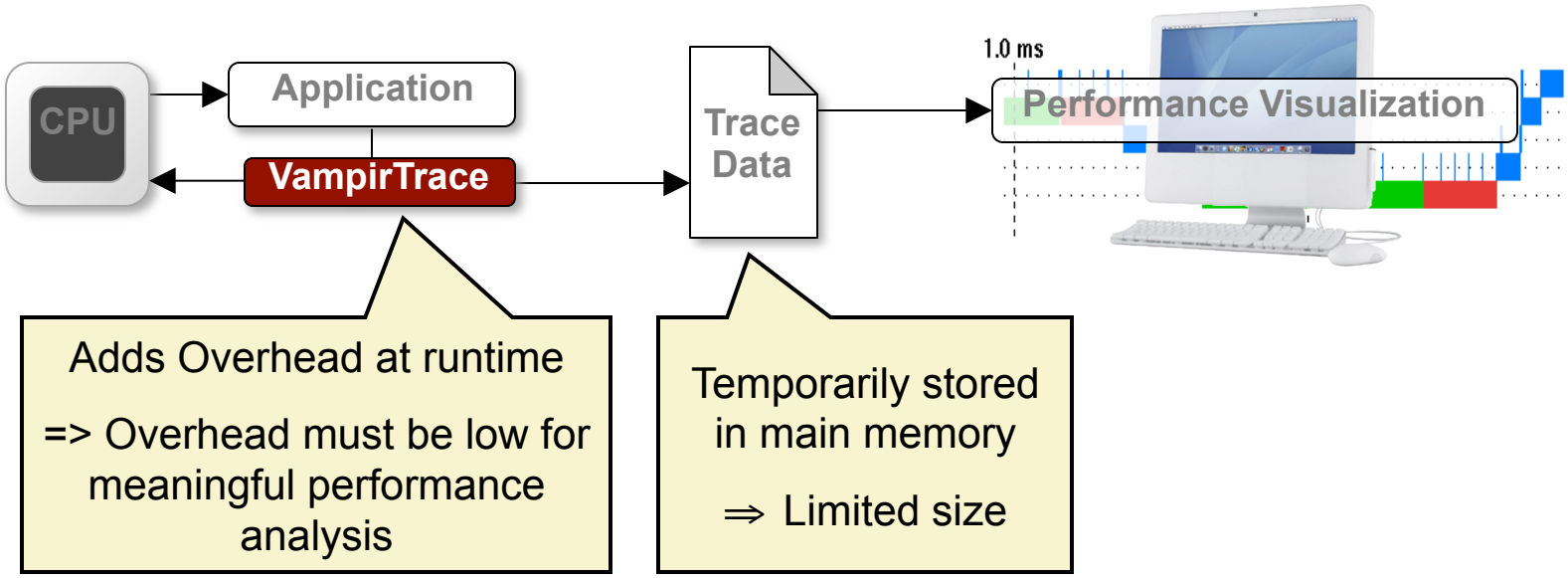
MPI Profiling Interface
- MPI

Runtime/Library
- MPI Correctness
- Pthreads
- NVIDIA CUDA
- OpenCL
- 3rd party libraries
- External counters
- Plugin counters
- Java tracing

Operating System
- Resource usage
- Memory allocation

Hardware
- Performance counters
- I/O
- CPU ID

Tracing – The Hard Part



- Event tracing requires trade-off's:
  - ⇒ Only add the data sources you need
  - ⇒ Limit granularity (i.e., filtering)
- Good practice: start with a profile run

Tracing – Usage

- Compiler wrappers provide instrumentation:

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

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

- Basic VampirTrace usage:
  - Exchange compiler commands
  - Re-compile & re-link
  - Run as usual (details follow)
- This adds compiler based instrumentation

Tracing – Example (1)

- NAS Parallel Benchmarks
  - Activate the VampirTrace compiler wrappers:

```
% vi config/make.def
            #F77 = mpif77
            F77 = vtf77 -vt:f77 mpif77
```

  - Build a kernel:

```
% make bt CLASS=W NPROCS=4
```

Tracing – Example (2)

- Remember: Start with a profile run
- Statistics run with VampirTrace:

```
% cd bin
% export VT_MODE=STAT
% mpirun -np 4 bt.W.4
…
 excl.  incl.          excl. time incl. time
 time   time    calls    /call     /call      ame
 2.867s 2.867s    2412  1.189ms   1.189ms  M  I_Wait
 1.060s 1.060s       1  1.060s    1.060s   M  I_Init
 0.765s 0.765s     202  3.788ms   3.788ms  MPI_Waitall
 0.483s 0.483s 1678149  0.287us   0.287us  binvcrhs_
 0.304s 0.304s 1678149  0.181us   0.181us  matmul_sub_
 0.298s 0.635s     402  0.741ms   1.578ms  x_solve_cell_
 0.297s 0.664s     402  0.740ms   1.651ms  z_solve_cell_
 0.264s 0.571s     402  0.656ms   1.419ms  y_solve_cell_
 0.210s 0.210s 1678149  0.125us   0.125us  matvec_sub_
 0.163s 0.163s     202  0.807ms   0.807ms  compute_rhs_
Displayed 10 from 65 functions.
```

Tracing very short but frequent calls is impractical

⇒ Candidates for filtering

Tracing – Example (3)

- Lets trace with filtering:

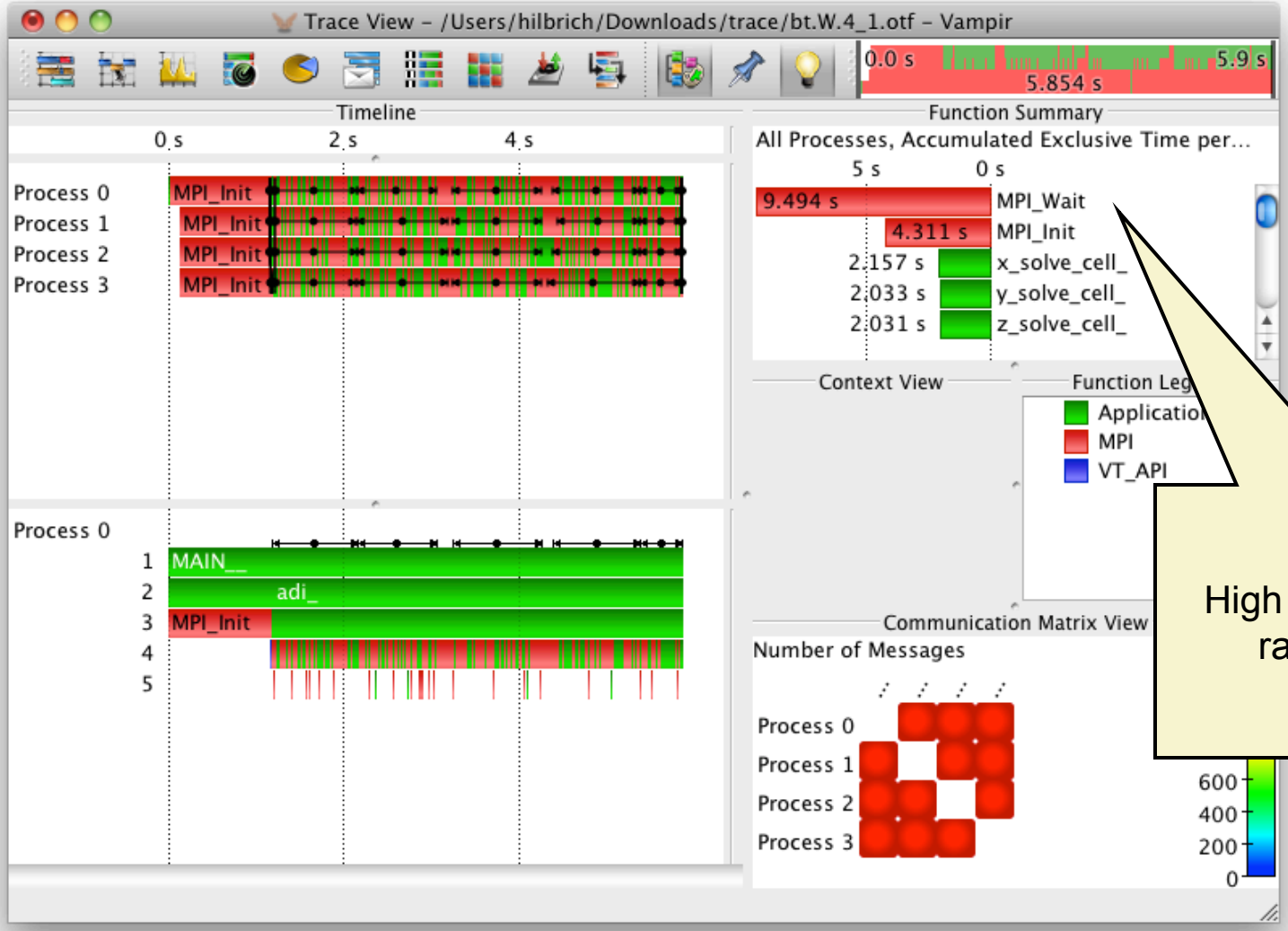
```
% nano filter.txt
binvcrhs_ -- 0
matmul_sub_ -- 0
matvec_sub_ -- 0
binvrhs_ -- 0
exact_solution_ -- 0
% export VT_MODE=TRACE
% export VT_FILTER_SPEC=filter.txt
% mpirun -np 4 bt.W.4

 NAS Parallel Benchmarks 3.3 -- BT Benchmark
 …
```

- Resulting trace file “bt.W.4.otf”:

```
% vampir bt.W.4.otf
```

Tracing – Example (4)



High MPI ratio due to 4 ranks on 2 cores

Tracing – Adding Detail

- We traced:
  - Function calls
  - MPI calls
- We might add (examples):
  - More buffering for the trace (for longer runs):

    ⇒
    ```
    % export VT_BUFFER_SIZE=1000M
    ```
  - PAPI performance counters:

    ⇒
    ```
    % export VT_METRICS=PAPI_FP_OPS:PAPI_L2_TCM
    ```
  - IO or memory allocation tracing:

    ⇒
    ```
    % export VT_MEMTRACE=yes
    % export VT_IOTRACE=yes
    ```

Tracing – Compiler Instrumentation Limits

- Default instrumentation: compiler instrumentation
  - Instruments EVERY function call
  - Even with filter: VampirTrace invoked per function call
  - ⇒ Can cause high overhead
- More advanced instrumentation techniques:
  - Manual (laborious, but always available)
  - Binary instrumentation with Dyninst
  - Source-to-source instrumentation with TAU PDT
- Advice: If compiler instrumentation causes high overhead, use one of the above

Tracing – Dyninst / PDT Instrumentation

- Instrumentation and execution with PDT:

Activates PDT instrumentation (Optional VampirTrace feature)

```
% vtcc -vt:cc mpicc -vt:inst tauinst \
      -vt:tau '-f tau_filter.txt' source.c -o exe
% mpirun -np 4 ./e
```

Sets a PDT filter (PDT specific format)

- Instrumentation and execution with Dyninst:

No modifications for compile&link necessary

```
% mpicc source.c -o exe -g
% mpirun -np 4 vtrun -mpi --dyninst ./exe
```

"vtrun" attaches VampirTrace and invokes dyninst instrumentation

Tracing – vtsetup

- A tool that aids in setting up VampirTrace runs

VTSetup

File Generate View Help

Configuration Steps:

- General Trace Settings
- Optional Trace Features
- Counter
- Filtering and Grouping
  - Step 1 General Setup
  - Step 2 Group Setup
  - Step 3 Filter Setup

Counter

-PAPI Performance Counter-

Name:	Derived:	Description:
PAPI_L1_DCM	No	Level 1 data cache misses
PAPI_L1_ICM	No	Level 1 instruction cache misses
PAPI_L2_DCM	Yes	Level 2 data cache misses
PAPI_L2_ICM	No	Level 2 instruction cache misses
PAPI_L1_TCM	No	Level 1 cache misses
PAPI_L2_TCM	No	Level 2 cache misses
PAPI_CA_SHR	No	Requests for exclusive access to sh...
PAPI_CA_CLN	No	Requests for exclusive access to cle...
PAPI_CA_ITV	No	Requests for cache line intervention
PAPI_TLB_DM	No	Data translation lookaside buffer mi...
PAPI_TLB_IM	No	Instruction translation lookaside buf...
PAPI_L1_LDM	No	Level 1 load misses
PAPI_L1_STM	No	Level 1 store misses
PAPI_L2_LDM	Yes	Level 2 load misses
PAPI_L2_STM	No	Level 2 store misses

-> <-

☑ Hide unavailable counters

-Resource Usage Counter-

☐ ru_utime	☐ ru_stime	☐ ru_maxrss	☐ ru_ixrss
☐ ru_idrss	☐ ru_isrss	☐ ru_minflt	☐ ru_majflt
☐ ru_nswap	☐ ru_inblock	☐ ru_oublock	☐ ru_msgsnd
☐ ru_msg...	☐ ru_nsignals	☐ ru_nvcsw	☐ ru_nivcsw

Resource usage update interval (ms): 100

Infobox:

PAPI Performance Counter:

Specify the counter that shell be recorded during the performance measurement.

Default: -

Variable: VT_METRICS

ion Services & High Performance Computing

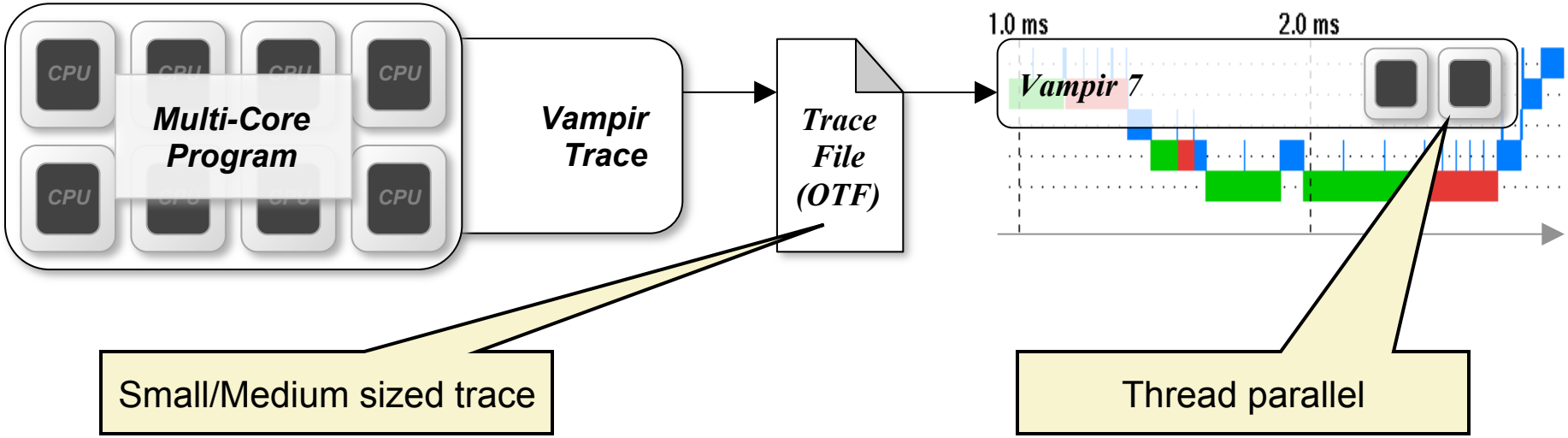
Content

- Overview
- Technology
- Example
- Tracing
- Visualization
- Meeting the Challenges
- Score-P
- Roadmap

Visualization – Vampir Modes (1)

- Directly on a local machine:

```
% vampir
```

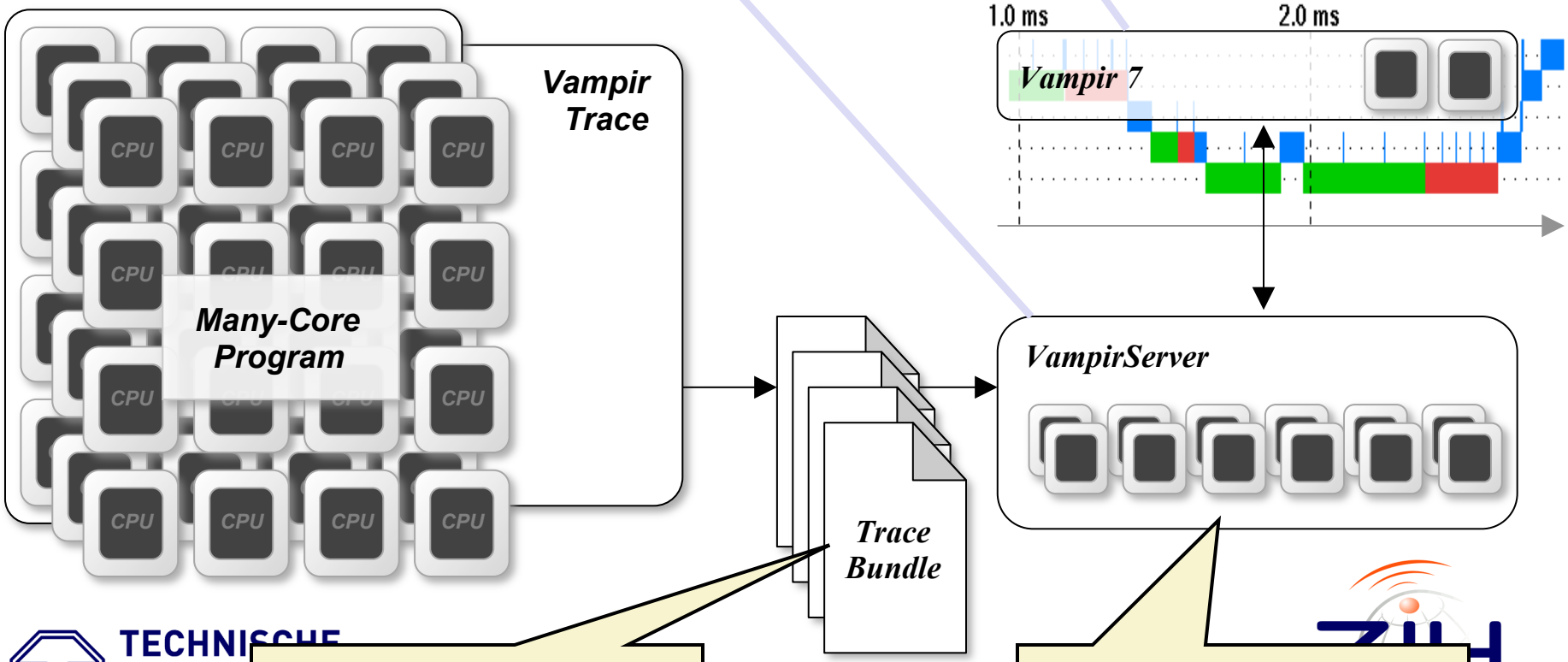


Visualization – Vampir Modes (2)

- On local machine with VampirServer:

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

```
% vampir
```

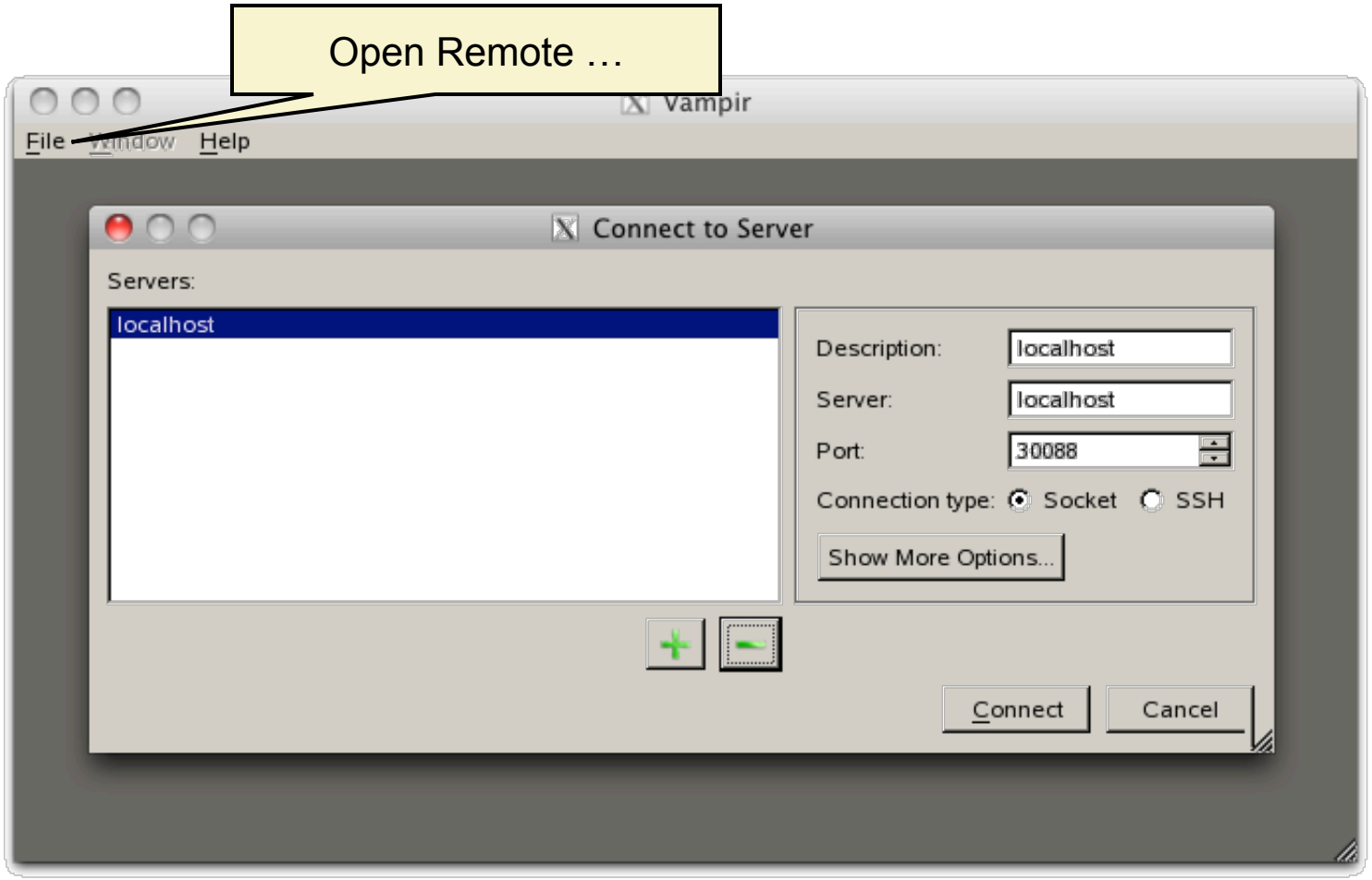


Large Trace File

MPI parallel application

Visualization – Vampir Modes (3)

- Connecting to VampirServer:



Visualization – Vampir Modes (4)

E.g. if X forwarding is too slow

- VampirServer on HPC system; GUI on desktop:

Desktop

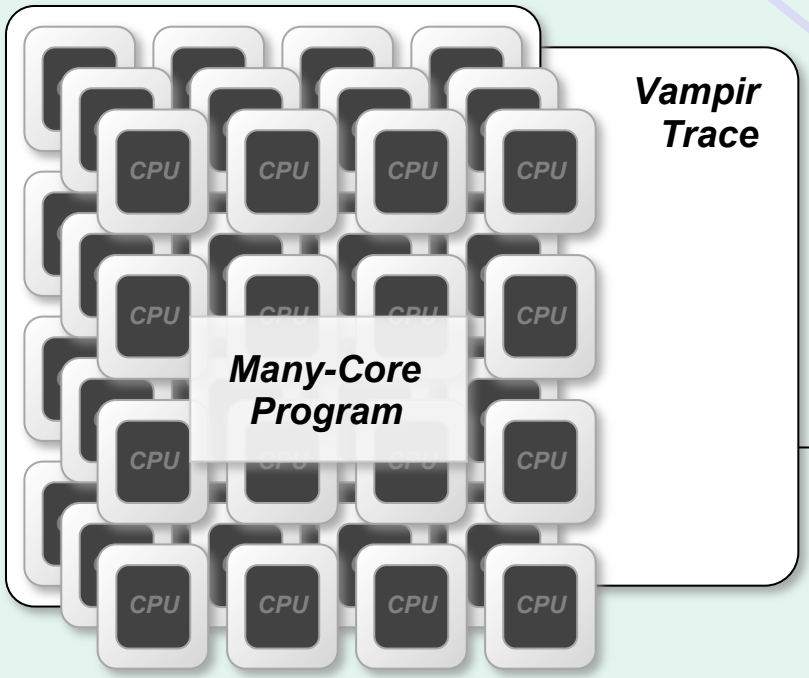
```
% ssh -L 30000:hpc-system:30088 \
      login@hpc-system
```

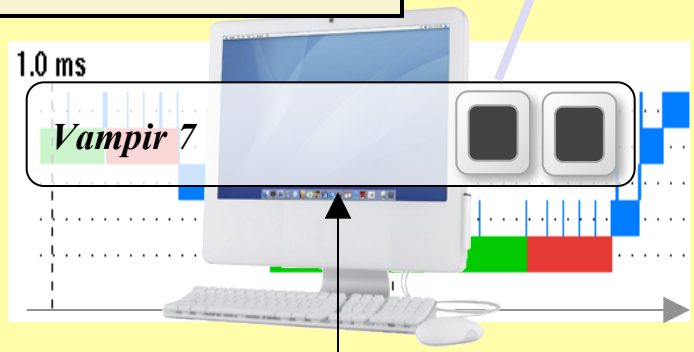
```
% vampir
```

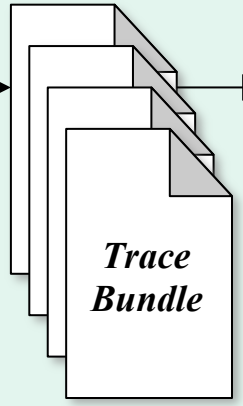
Forward local port 30000 to VampirServer on HPC system

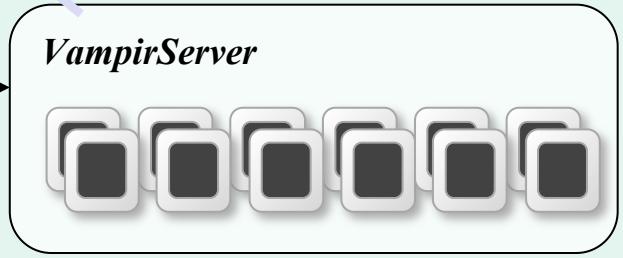
HPC system

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









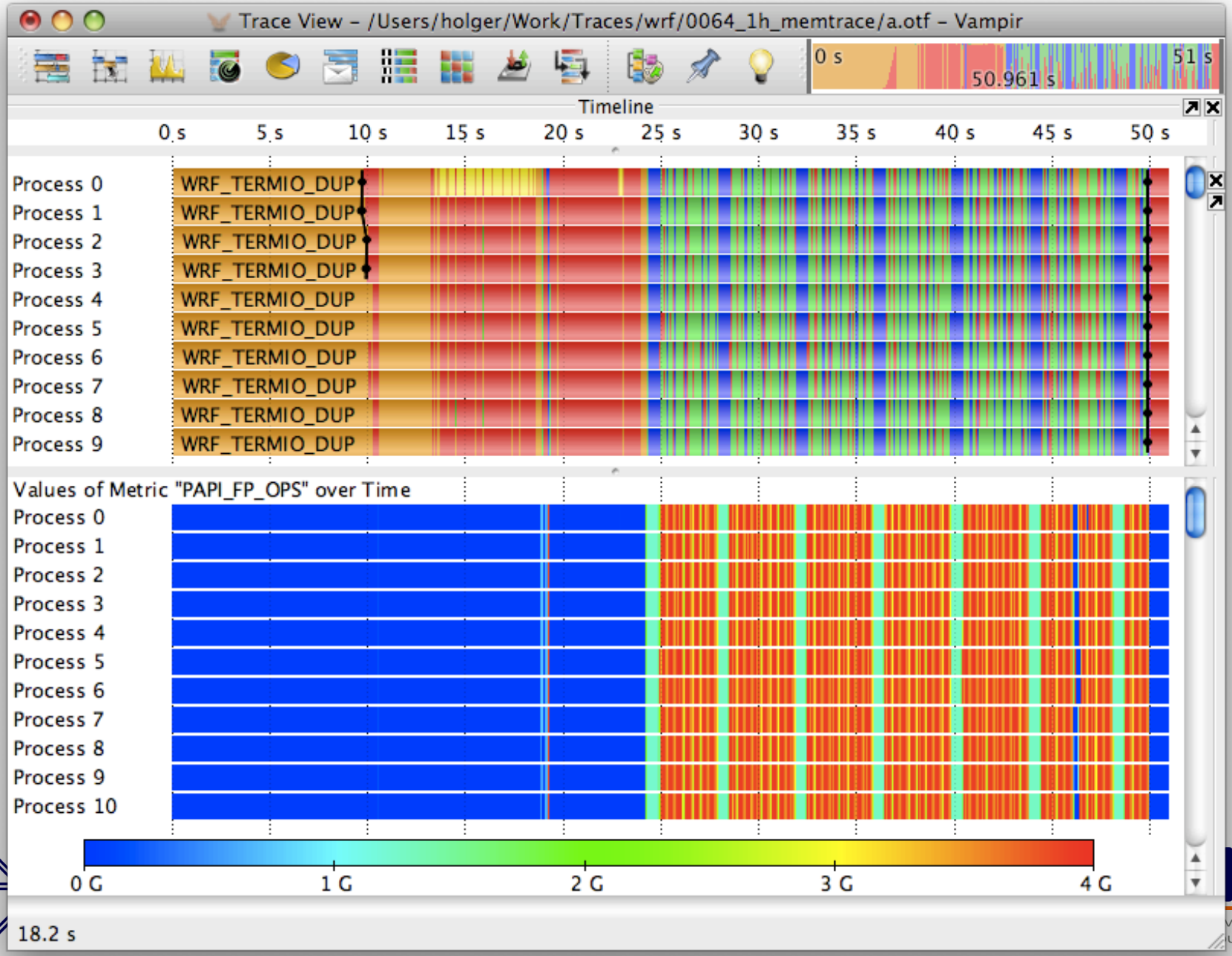
Content

- Overview
- Technology
- Example
- Tracing
- Visualization
- Meeting the Challenges
- Score-P
- Roadmap

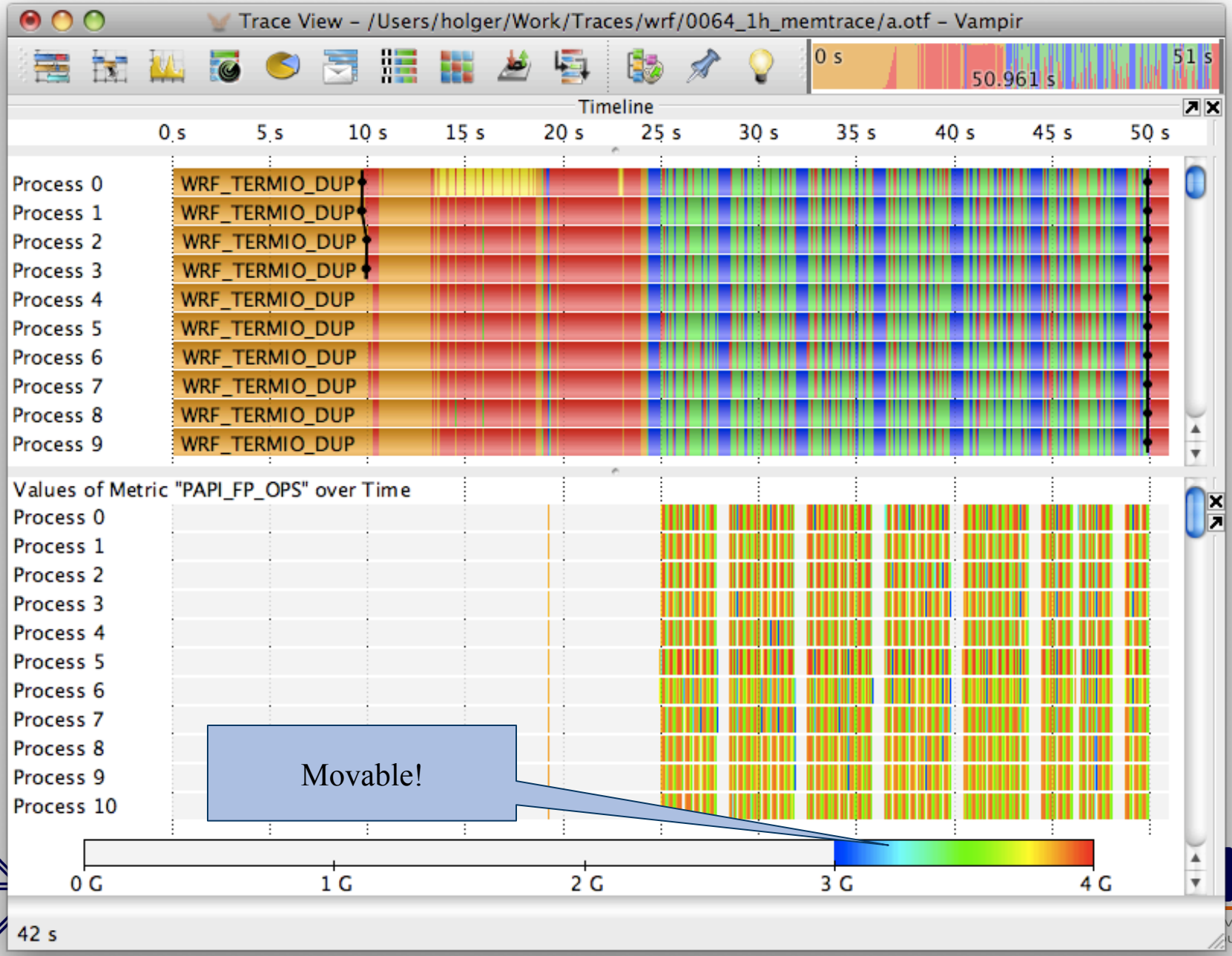
Meeting the Challenges

- HPC systems evolve; tools need adapt:
  - Scalability
  - Paradigms
  - Heterogeneity
  - Fault tolerance
  - Usability
- Several extensions address these challanges

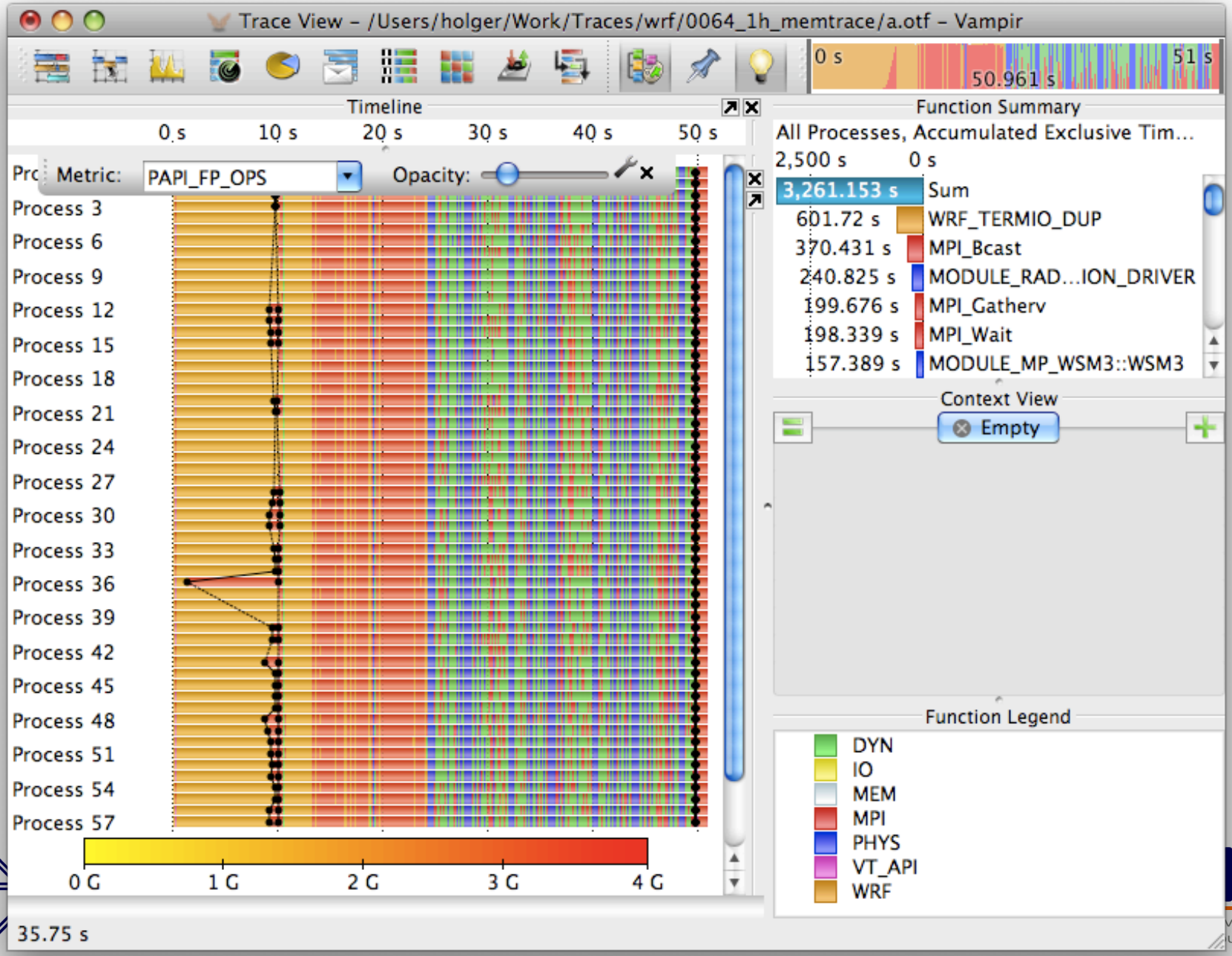
Meeting the Challenges – Performance Radar



Meeting the Challenges – Performance Radar



Meeting the Challenges – Overlays



Meeting the Challenges – Overlays

Trace View – /Users/holger/Work/Traces/wrf/0064_1h_memtrace/a.otf – Vampir

0 s 50.961 s 51 s

Timeline

0 s 10 s 20 s 30 s 40 s 50 s

Metric: PAPI_FP_OPS Opacity:

Process 3
Process 6
Process 9
Process 12
Process 15
Process 18
Process 21
Process 24
Process 27
Process 30
Process 33
Process 36
Process 39
Process 42
Process 45
Process 48
Process 51
Process 54
Process 57

0 G 1 G 2 G 3 G 4 G

35.75 s

Function Summary

All Processes, Accumulated Exclusive Tim...

2,500 s	0 s
3,261.153 s	Sum
601.72 s	WRF_TERMIO_DUP
370.431 s	MPI_Bcast
240.825 s	MODULE_RAD...ION_DRIVER
199.676 s	MPI_Gatherv
198.339 s	MPI_Wait
157.389 s	MODULE_MP_WSM3::WSM3

Context View

Empty

Function Legend

DYN
IO
MEM
MPI
PHYS
VT_API
WRF

Meeting the Challenges – Overlays

Trace View – /Users/holger/Work/Traces/wrf/0064_1h_memtrace/a.otf – Vampir

0 s 50.961 s 51 s

Timeline

0 s 10 s 20 s 30 s 40 s 50 s

Metric: PAPI_FP_OPS Opacity:

Process 3
Process 6
Process 9
Process 12
Process 15
Process 18
Process 21
Process 24
Process 27
Process 30
Process 33
Process 36
Process 39
Process 42
Process 45
Process 48
Process 51
Process 54
Process 57

0 G 1 G 2 G 3 G 4 G

37 s

Function Summary

All Processes, Accumulated Exclusive Tim...

2,500 s	0 s
3,261.153 s	Sum
601.72 s	WRF_TERMIO_DUP
370.431 s	MPI_Bcast
240.825 s	MODULE_RAD...ION_DRIVER
199.676 s	MPI_Gatherv
198.339 s	MPI_Wait
157.389 s	MODULE_MP_WSM3::WSM3

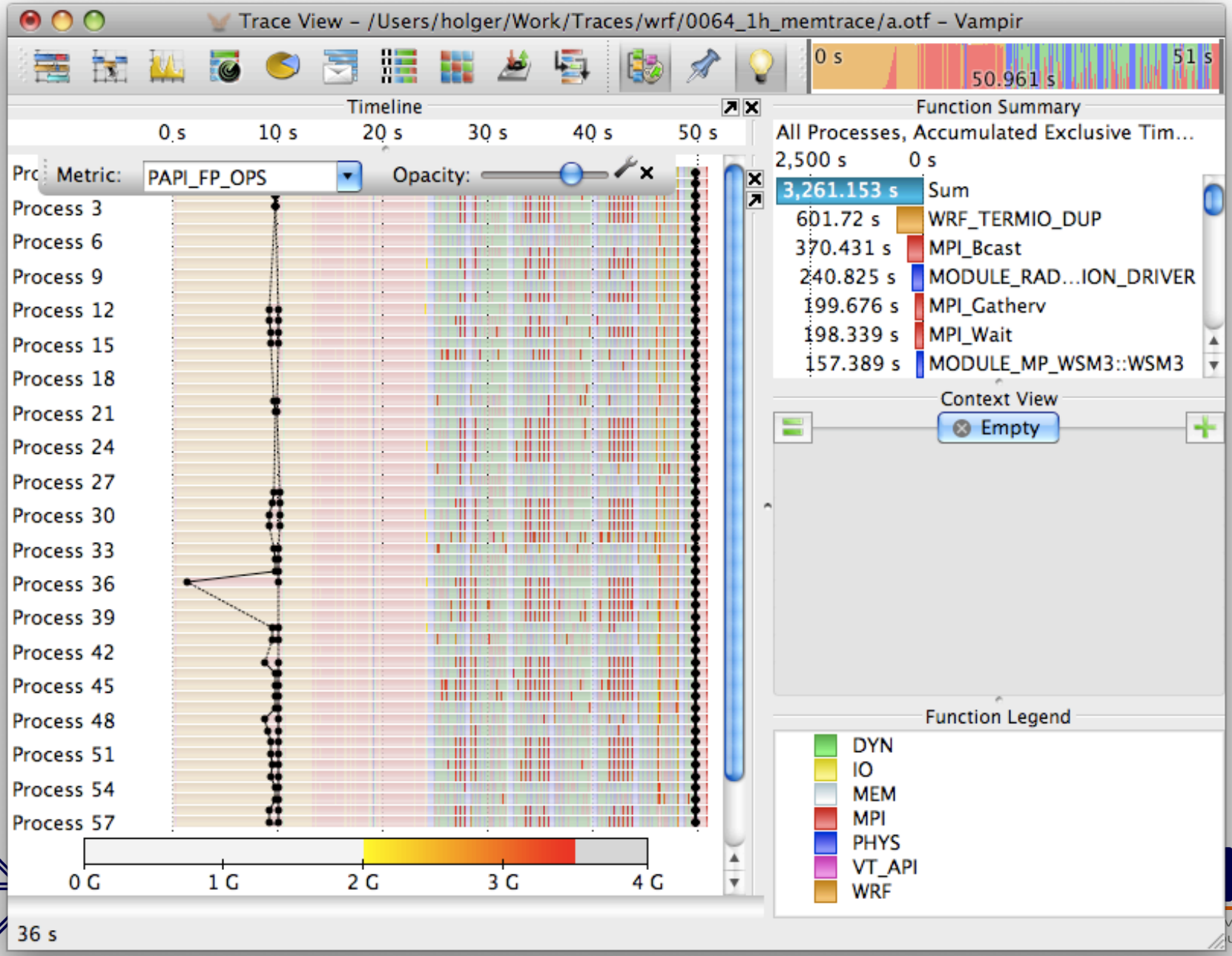
Context View

Empty

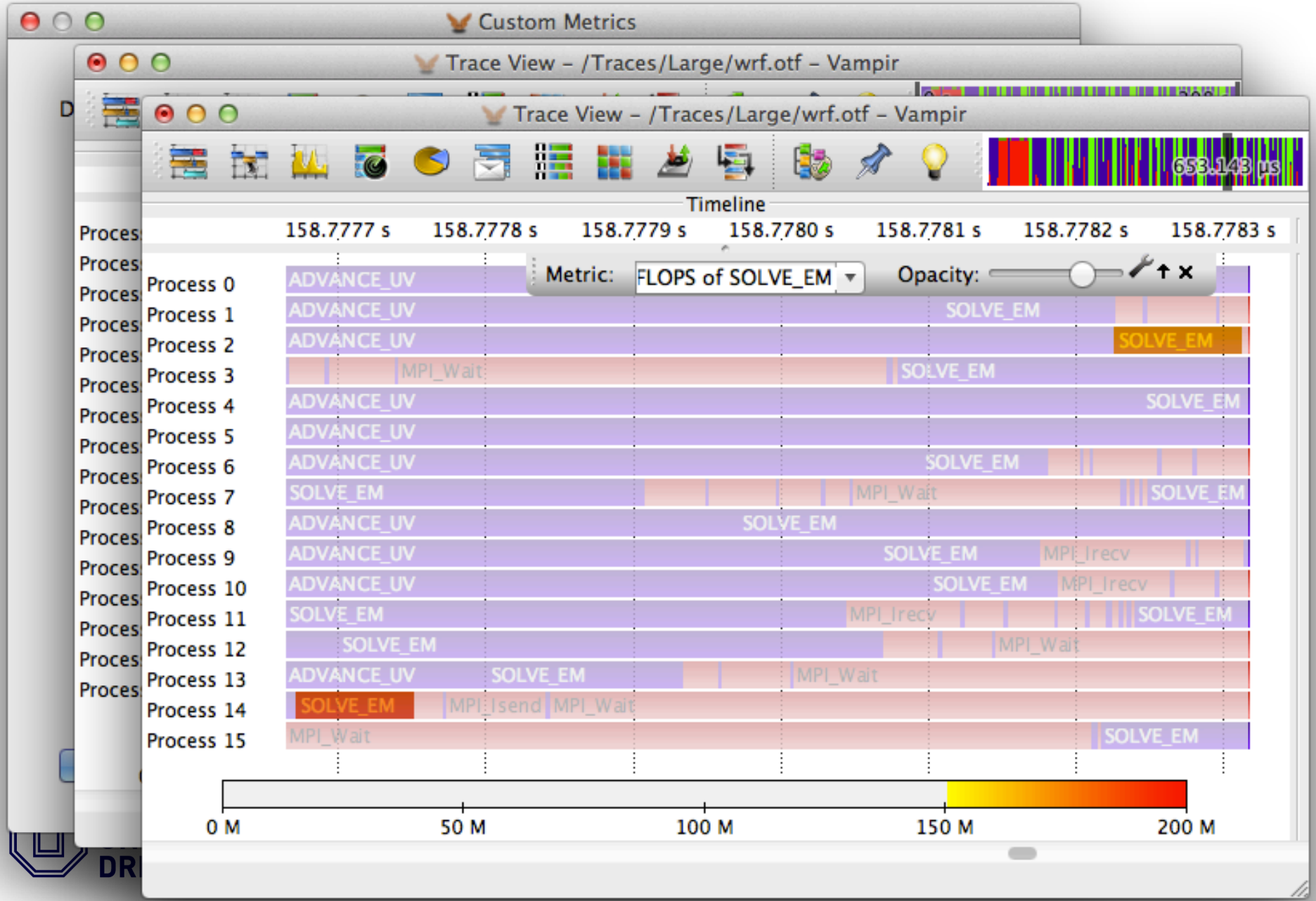
Function Legend

DYN
IO
MEM
MPI
PHYS
VT_API
WRF

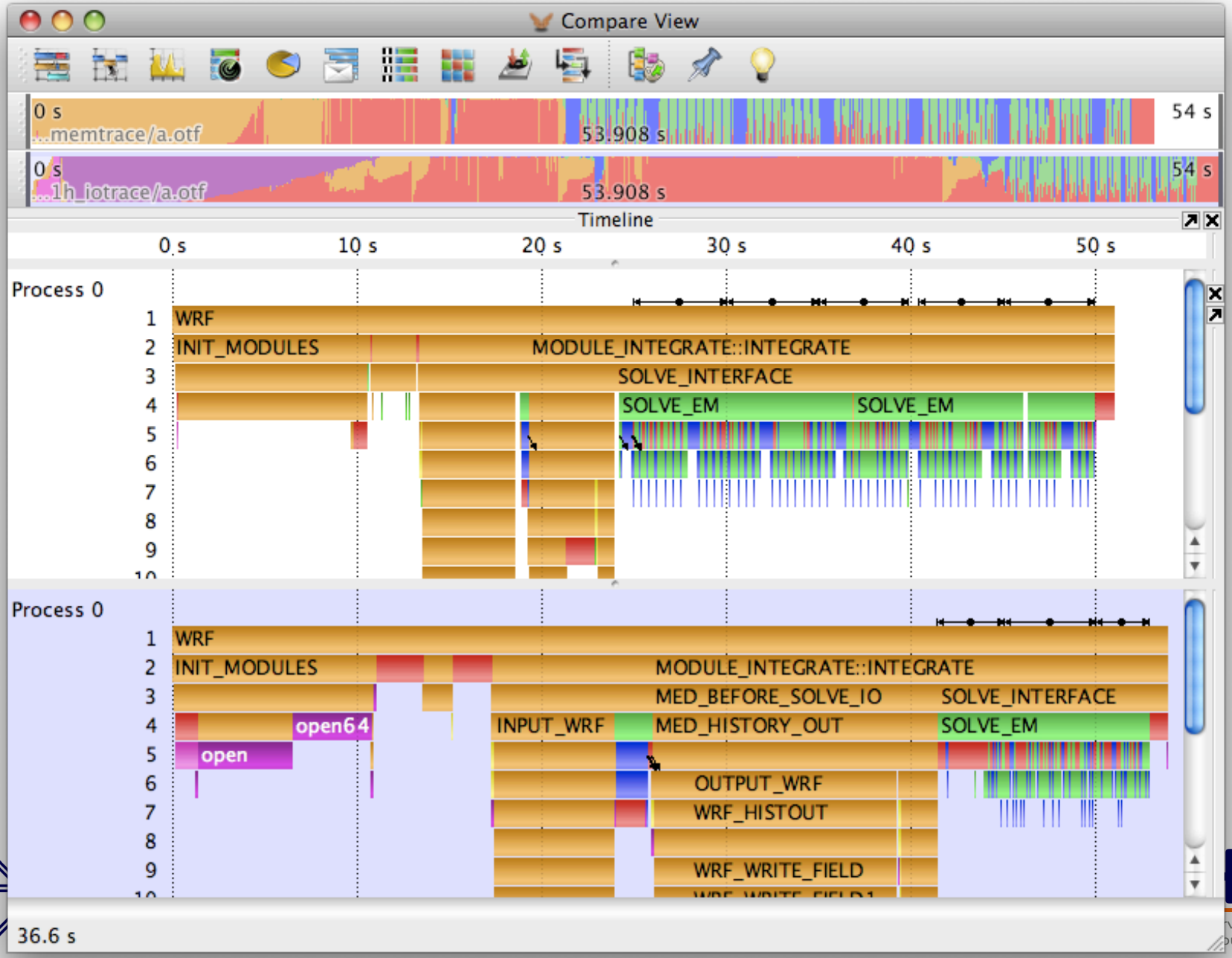
Meeting the Challenges – Overlays



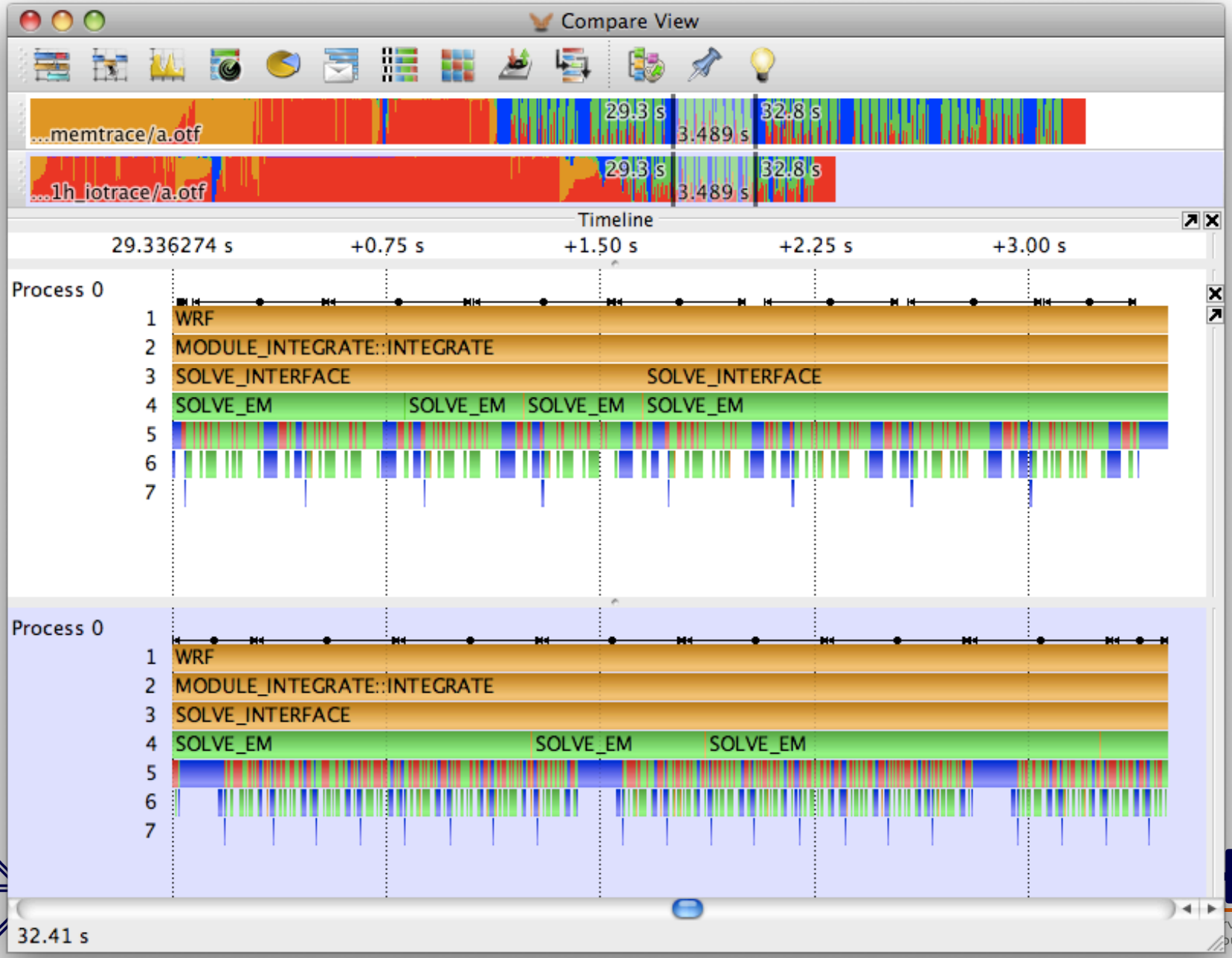
Meeting the Challenges – Derived Counters



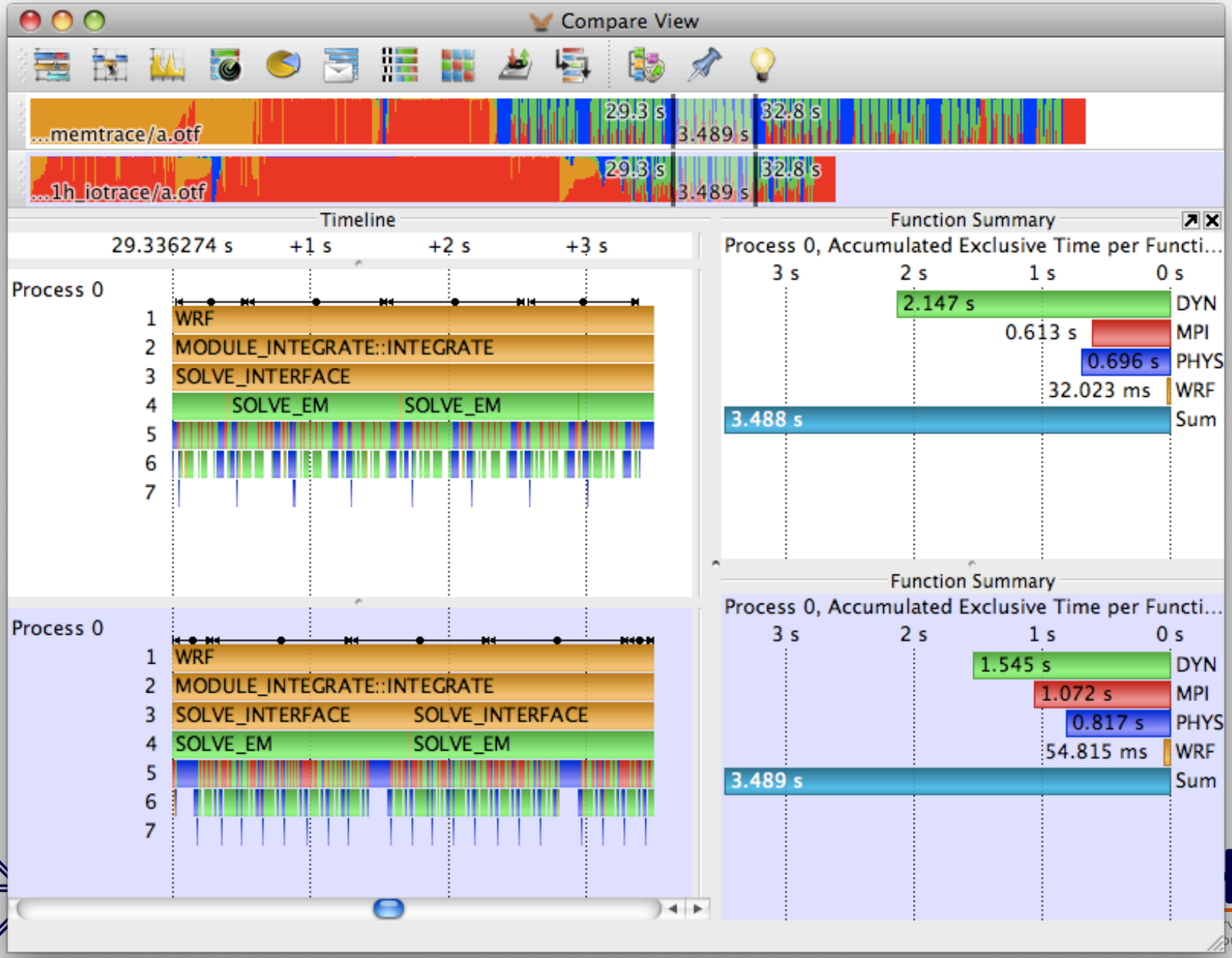
Meeting the Challenges – Compare View



Meeting the Challenges – Compare View



Meeting the Challenges – Compare View

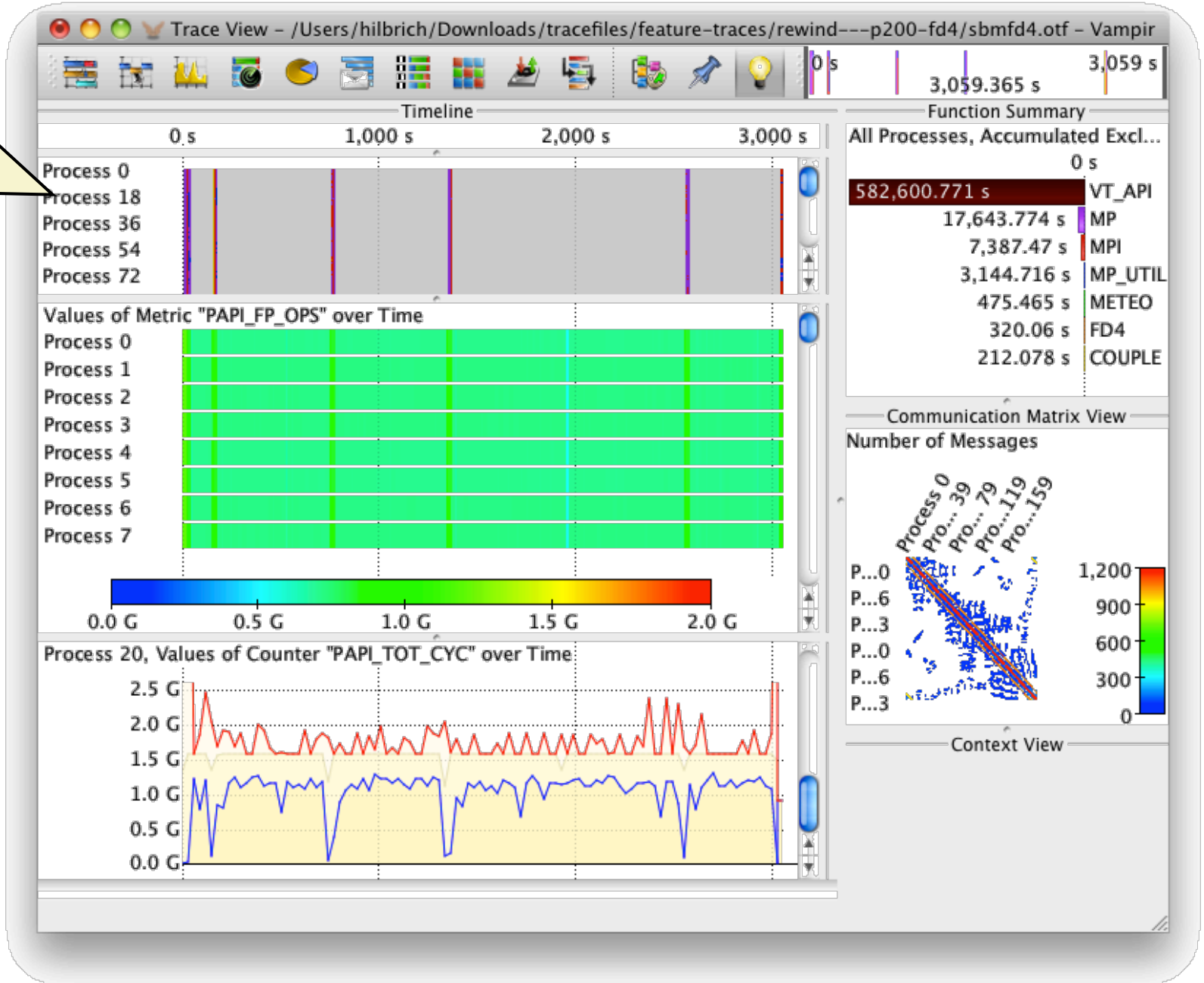


Meeting the Challenges – Long Tracing Runs (1)

- Tracing long running applications is hard
- Case:
  - Long running application (~50min)
  - Non-reproducible bad performance in iterations
  - Coarse-grained traces gave no insight
  - Fine-grained traces would not fit into memory
- Solution:
  - VampirTrace feature “Rewind”
  - Allows to discard parts of the trace buffer
  - Controlled via VampirTrace instrumentation API

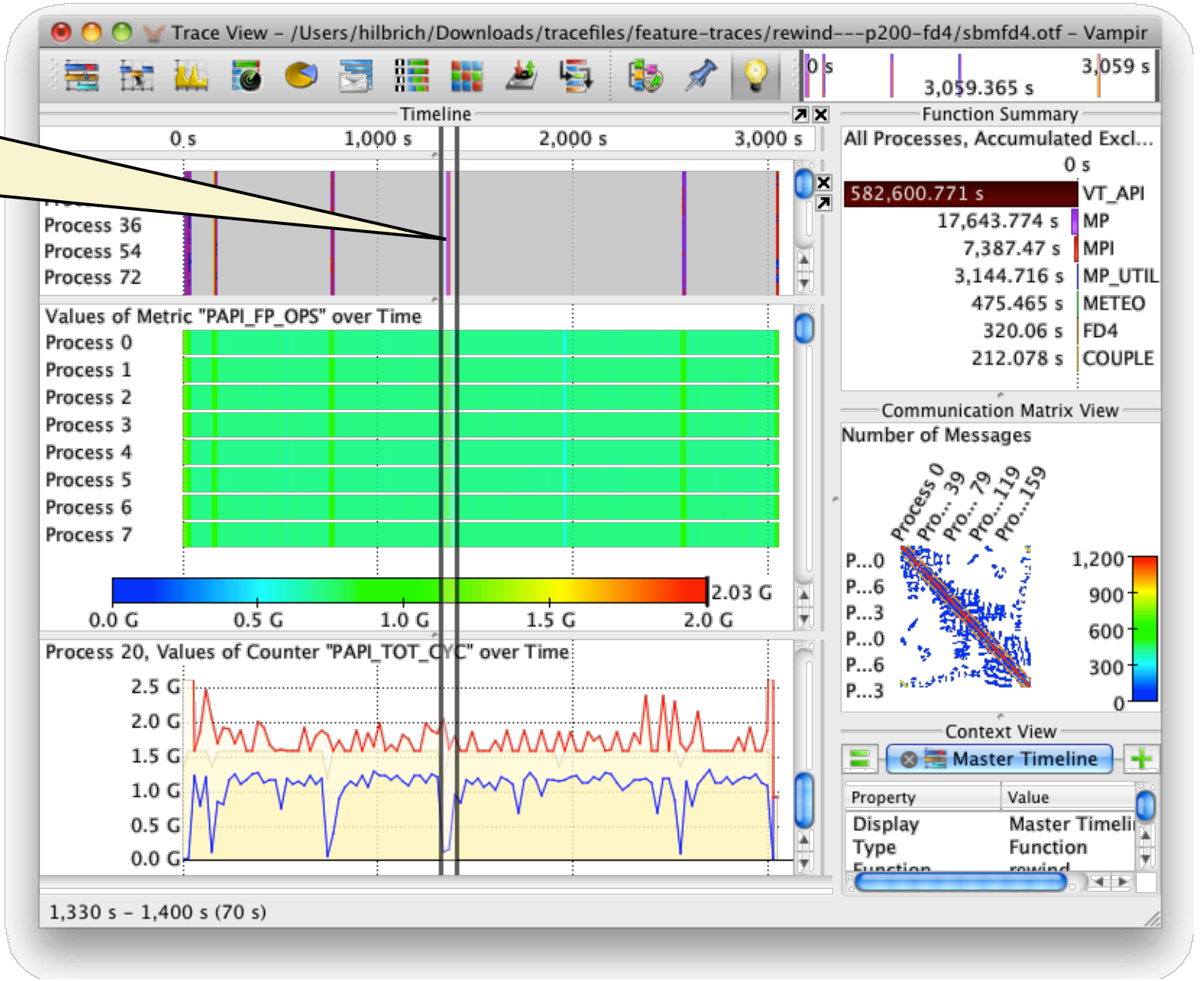
Meeting the Challenges – Long Tracing Runs (2)

First and last iterations
+ 4 “slow” iterations



Meeting the Challenges – Long Tracing Runs (3)

Zoom into a slow iteration

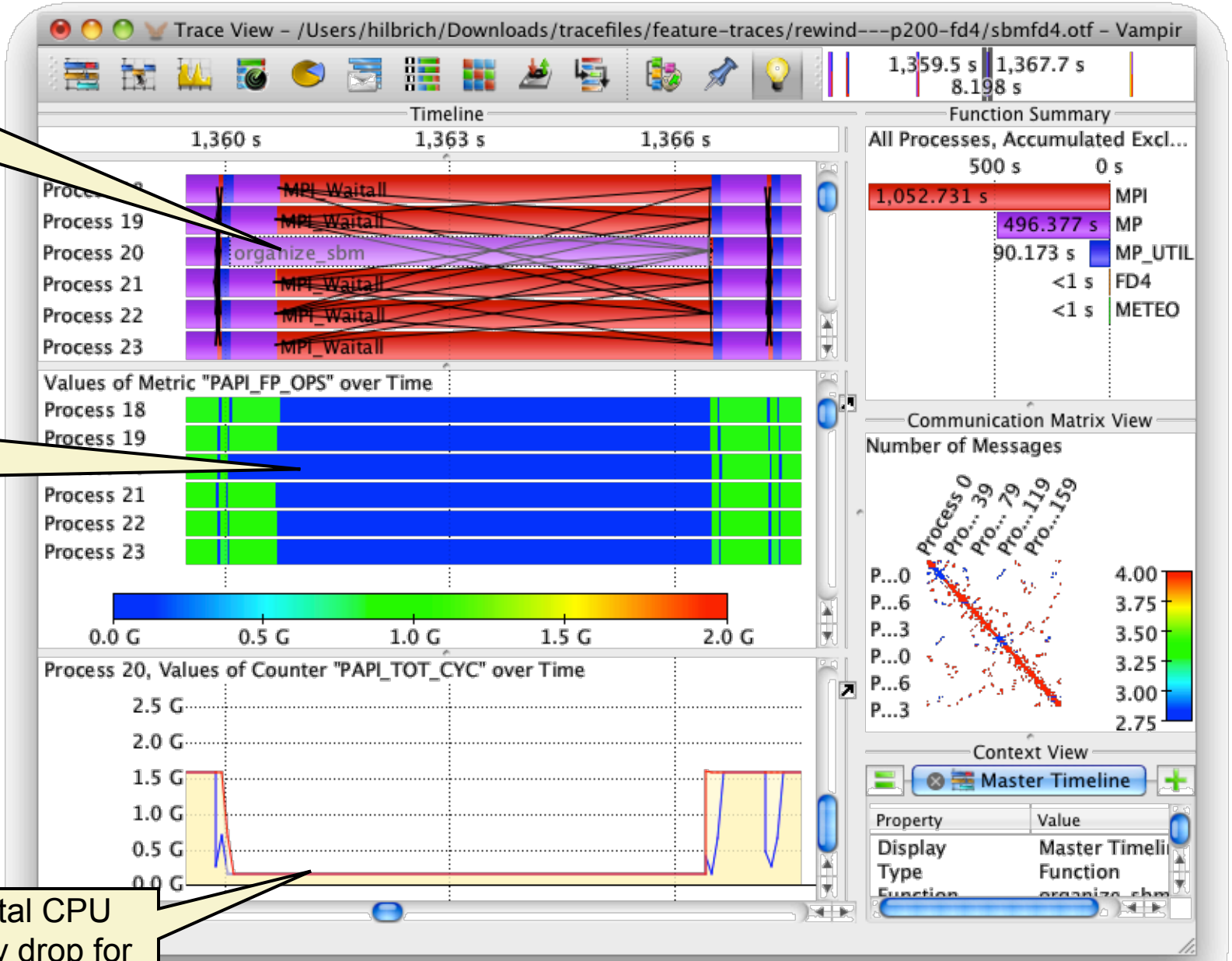


Meeting the Challenges – Long Tracing Runs (4)

All except rank 20 need to wait

Can also be spotted in Radar

Process 20 total CPU cycles suddenly drop for > 6 seconds



Meeting the Challenges – CUDA, PicOnGPU

- Case:
  - Particle in cell code on NVIDIA GPGPUs
  - Multi-level parallelism: MPI + CUDA + pthreads
  - Performance is a black box without tools
- Solution:
  - VampirTrace support for CUDA
  - Close collaboration with NVIDIA

Meeting the Challenges – CUDA, PicOnGPU

Trace View – /Users/hilbrich/Downloads/tracefiles/picongpu_CUDA_MPI_PTHREADS_OMP/trace.otf – Vampir

15 s 31 s 15.875 s

Timeline

20 s 25 s 30 s

Process 0
CUDA[0] 0:1
CUDA[0] 0:2
CUDA[0] 0:3
CUDA[0] 0:4
CUDA[0] 0:5
CUDA[0] 0:6
CUDA[0] 0:7
CUDA[0] 0:8
Thread 0:9
Thread 0:10
Thread 0:11
Thread 0:12

Function Summary

All Processes, Accumulated Exclusive Time pe...

100 s 0 s

167.925 s	Application
45.834 s	CUDA_KERNEL
17.667 s	CUDA_IDLE
1.479 s	MPI
<1 s	CUDART_API
<0.1 s	VT_CUDA
<100 µs	CUDA_SYNC

Context View

Function Legend

- Application
- CUDART_API
- CUDA_IDLE
- CUDA_KERNEL
- CUDA_SYNC
- MPI
- VT_CUDA

Meeting the Challenges – CUDA, PicOnGPU

Trace View – /Users/hilbrich/Downloads/tracefiles/picongpu_CUDA_MPI_PTHREADS_OMP/trace.otf – Vampir

15 s 31 s 15.875 s

Timeline

20 s 25 s 30 s

Process 0
CUDA[0] 0:3
CUDA[0] 0:6
Thread 0:9
Thread 0:12
CUDA[1] 1:2
CUDA[1] 1:5
CUDA[1] 1:8
Thread 1:11
CUDA[2] 2:1
CUDA[2] 2:4
CUDA[2] 2:7
Thread 2:12
Process 3
CUDA[3] 3:3
CUDA[3] 3:6
Thread 3:9

Function Summary

All Processes, Accumulated Exclusive Time pe...

100 s 0 s

167.925 s	Application
45.834 s	CUDA_KERNEL
17.667 s	CUDA_IDLE
1.479 s	MPI
<1 s	CUDART_API
<0.1 s	VT_CUDA
<100 µs	CUDA_SYNC

Context View

Function Legend

- Application
- CUDART_API
- CUDA_IDLE
- CUDA_KERNEL
- CUDA_SYNC
- MPI
- VT_CUDA

Meeting the Challenges – CUDA, PicOnGPU

Trace View – /Users/hilbrich/Downloads/tracefiles/picongpu_CUDA_MPI_PTHREADS_OMP/trace.otf – Vampir

15 s 31 s 15.875 s

Timeline | Function Summary

Filter Processes

Process Groups
Communicators
Process Hierarchy

Number of processes
52

Selected processes
32

Include/Exclude All	
GPU_COMM_GLOBAL	28/32 >
GPU_GROUP	32/32 >
MPI Communicator 0	0/4 >
MPI_COMM_SELF 0	0/1 >
MPI_COMM_SELF 1	0/1 >
MPI_COMM_SELF 2	0/1 >
MPI_COMM_SELF 3	0/1 >
MPI_COMM_WORLD	0/4 >
n01	32/52 >

Include/Exclude All
CUDA[1] 1:8
CUDA[2] 2:1
CUDA[2] 2:2
CUDA[2] 2:3
CUDA[2] 2:4
CUDA[2] 2:5
CUDA[2] 2:6
CUDA[2] 2:7
CUDA[2] 2:8
CUDA[3] 3:1
CUDA[3] 3:2
CUDA[3] 3:3
CUDA[3] 3:4
CUDA[3] 3:5
CUDA[3] 3:6
CUDA[3] 3:7
CUDA[3] 3:8

Reset Apply Cancel OK

Process 0
CUDA[0] 0:3
CUDA[0] 0:6
Thread 0:9
Thread 0:12
CUDA[1] 1:2
CUDA[1] 1:5
CUDA[1] 1:8
Thread 1:11
CUDA[2] 2:1
CUDA[2] 2:4
CUDA[2] 2:7
Thread 2:12
Process 3
CUDA[3] 3:3
CUDA[3] 3:6
Thread 3:9

Meeting the Challenges – CUDA, PicOnGPU

Trace View – /Users/hilbrich/Downloads/tracefiles/picongpu_CUDA_MPI_PTHREADS_OMP/trace.otf – Vampir

15 s 31 s 15.875 s

Timeline

25 s

CUDA[0] 0:1
CUDA[0] 0:3
CUDA[0] 0:5
CUDA[0] 0:7
CUDA[1] 1:1
CUDA[1] 1:3
CUDA[1] 1:5
CUDA[1] 1:7
CUDA[2] 2:1
CUDA[2] 2:3
CUDA[2] 2:5
CUDA[2] 2:7
CUDA[3] 3:1
CUDA[3] 3:3
CUDA[3] 3:5
CUDA[3] 3:7

Function Summary

All Processes, Accumulated Exclusive Time per Function

10 s 0 s

17.667 s	gpu_idle
12.071 s	picongpuFra...ILj8,j8,j4>
9.315 s	picongpuFra...rkParticles
7.08 s	picongpuFr...sorbBorder
5.843 s	picongpuFr...nelUpdateB
5.092 s	picongpuFr...nelUpdateE
3.247 s	picongpuFr...CurrentToE
1.503 s	GPUGrid::kernelSetValue
0.432 s	GridSimulati...rnelFillGaps

Context View

Function Su...

Value

Function Summary
picongpuFrame::kernelComp
CUDA_KERNEL (5)
12.07145 s (19.01%)

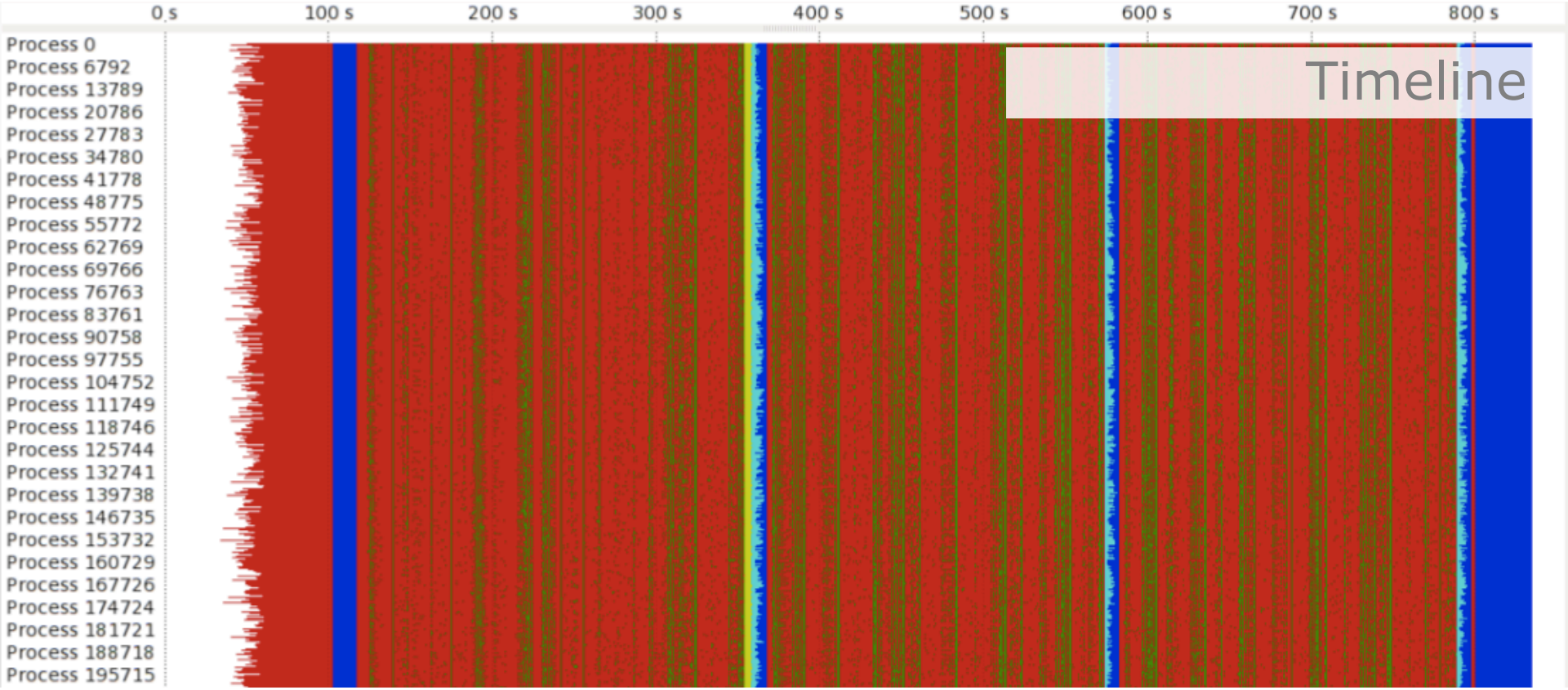
Function Legend

- Application
- CUDART_API
- CUDA_IDLE
- CUDA_KERNEL
- MPI
- VT_CUDA

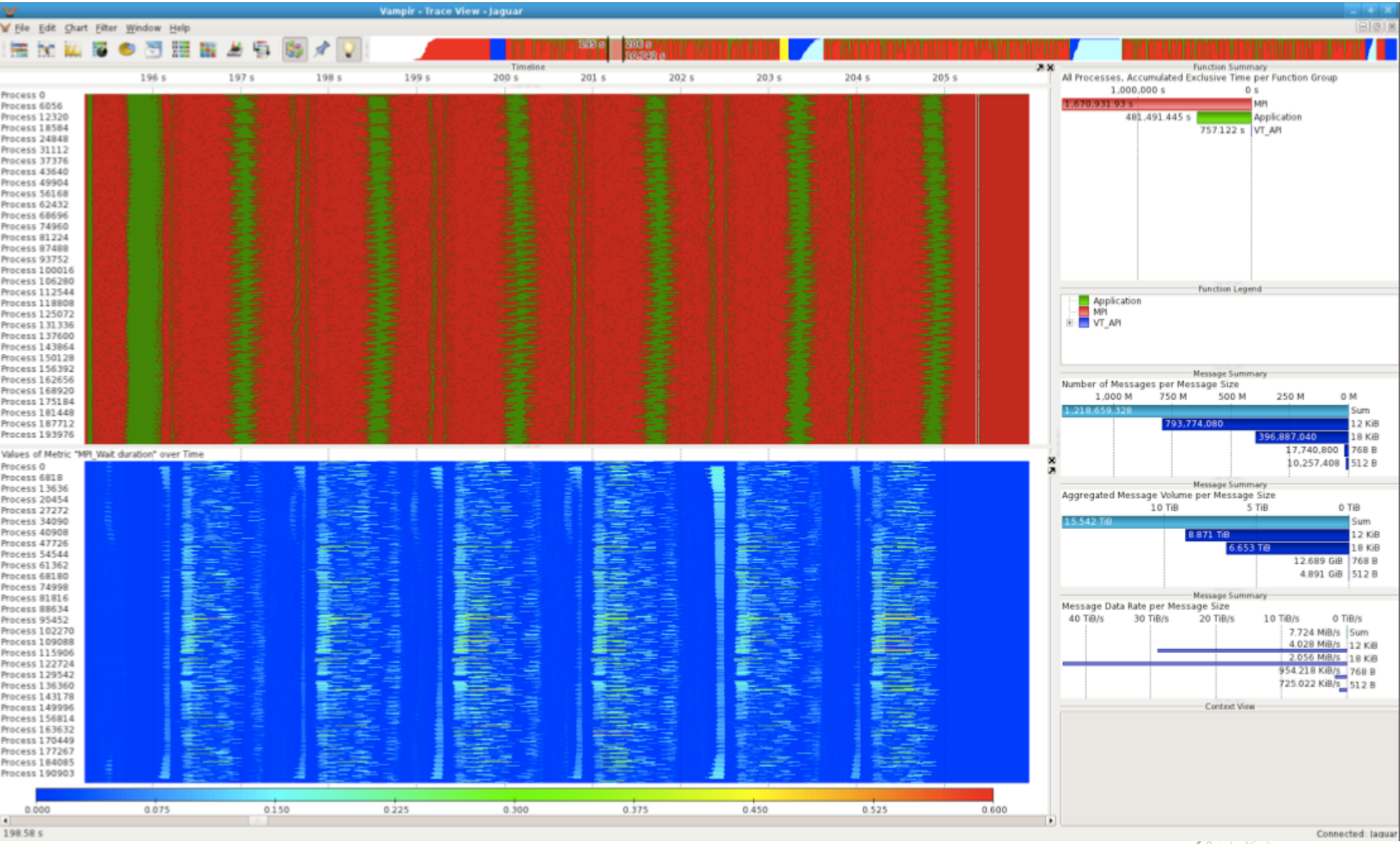
Meeting the Challenges – Collaboration with ORNL

- Goal:
  - Support hybrid system and software architecture at 10 Petascale
  - Show MPI and GPGPU programming
  - Do full system performance profiling **and** tracing
- Facts:
  - Jaguar + Successor
  - >220.000 cores
  - 200,448 monitored MPI processes
  - >20 Tera-bytes of performance data
  - 21,515 VampirServer processes

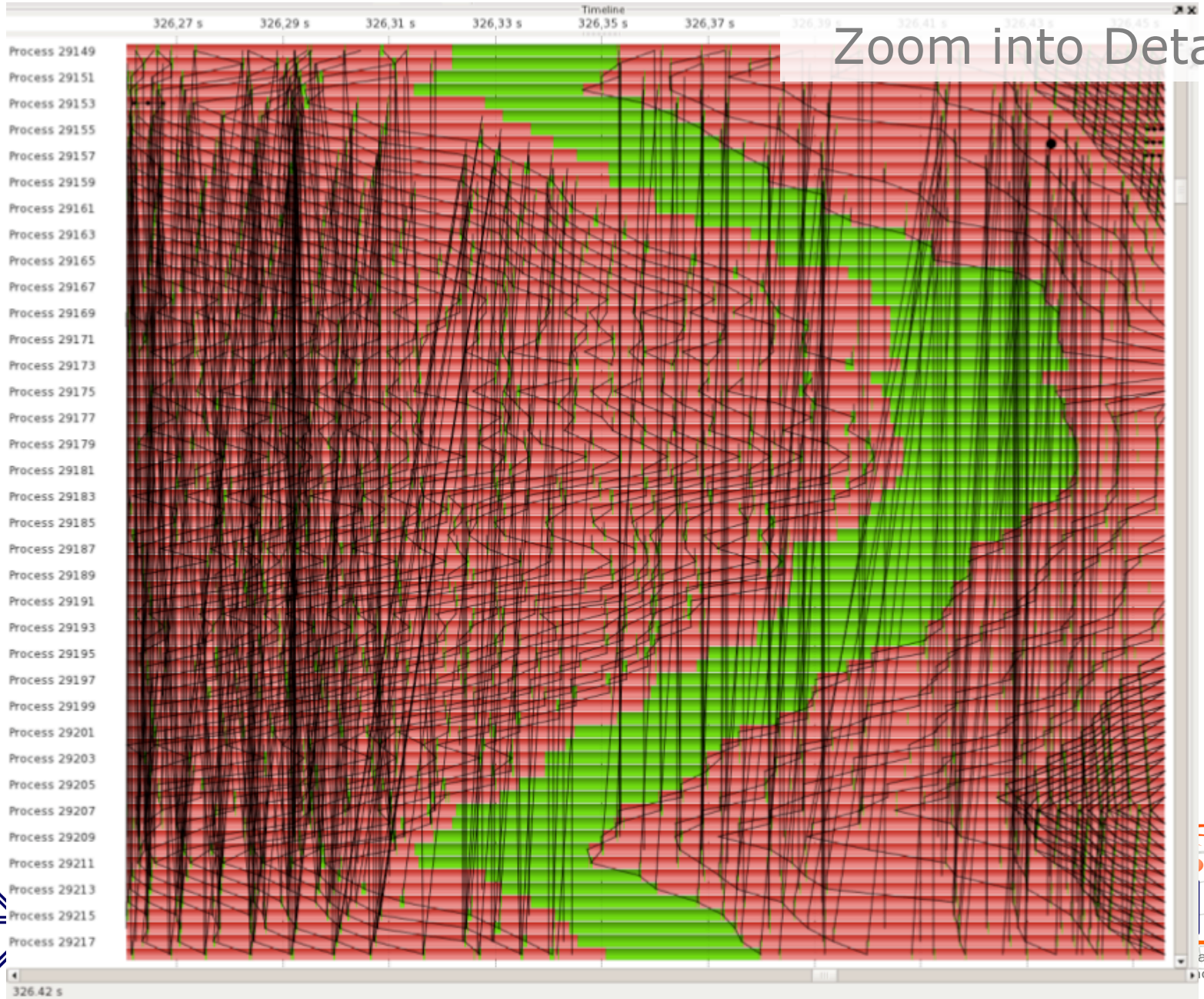
Meeting the Challenges – Collaboration with ORNL



Meeting the Challenges – Collaboration with ORNL



Meeting the Challenges – Collaboration with ORNL



Meeting the Challenges – Vampir I/O Analysis

- recording of POSIX I/O and MPI I/O operations with VampirTrace
- embedding of node local data like InfiniBand statistics
- on demand inclusion of external performance data from:
  - I/O network
  - storage controller
  - file servers
- within Vampir:
  - counter timelines for the host based and the external data
  - specialized I/O display to show:
    - details for single I/O events
    - grouping of events for the current portion of the timeline based on the filename, the type of the I/O record (read, write, …) and some more
    - I/O request size statistics

Meeting the Challenges – Vampir I/O Analysis

Vampir - [Trace View - /home/mkluge/threader_scaling_trace/dios_mpi_23.otf]

File Edit Chart Filter Window Help

0 s 604.79 s 605 s

Timeline

100 s 200 s 300 s 400 s 500 s 600 s

Process 28 Process 29 Process 30 Process 31 Process 32

MPI_Barrier MPI_Barrier MPI_Barrier

ExtCounter 0:1, Values of Counter "MDS001 remote procedure calls/s" over Time

10 k 5 k 0 k

ExtCounter 0:1, Values of Counter "pc_san: read bandwidth" over Time

44.25 M 43.50 M 42.75 M 42.00 M

ExtCounter 0:1, Values of Counter "mds001/load_1" over Time

4 k 3 k 2 k 1 k

Function Summary

All Processes, Accumulated Exclusiv...

10,000 s

31,116.835 s	fopen64
29,824.819 s	fclose
12,937.123 s	MPI_Barrier
10,830.98 s	fread

Function Legend

Application
LIBC-I/O
MPI
VT_API

I/O Summary

All Processes, Read, Sum of Size per File

0 GiB

73.626 GiB	Sum
14.305 MiB	/fastfs/mkluge/threader_database_single/164
14.301 MiB	/fastfs/mkluge/threader_database_single/3768
14.301 MiB	/fastfs/mkluge/threader_database_single/2696

Context View

Master Timeline

Property	Value
Display	Master Timeline
Type	I/O Event
Initiator	Process 5
File Name	/fastfs/mkluge/threader_database_single/3234
File Group	fileio
Operation	READ
Interval Begin	252.270591 s
Interval End	252.335219 s
Duration	64.6285 ms
Size	21.536133 KiB
Data Rate	333.229654 KiB/s

metadata rate

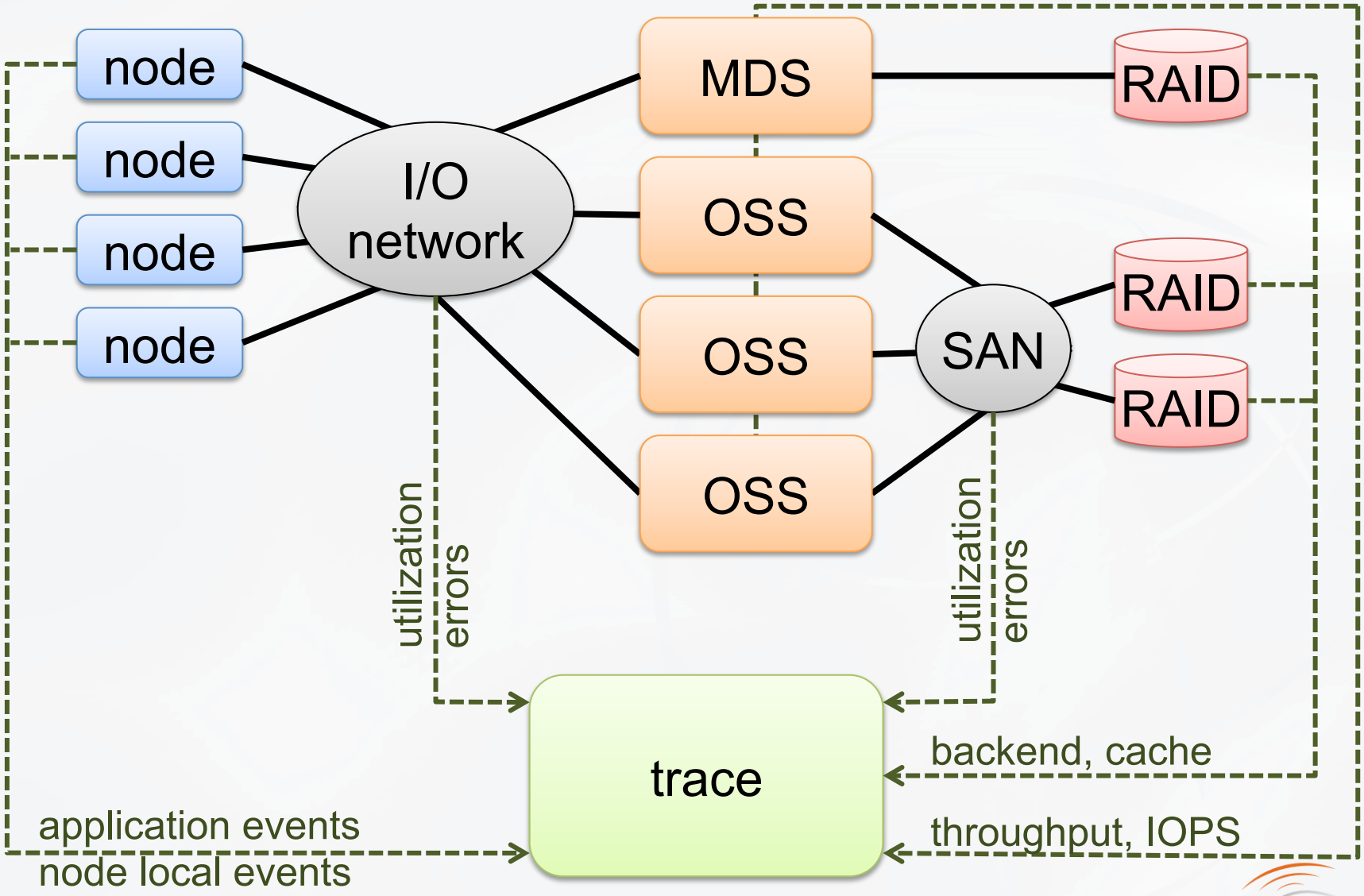
metadata server load

metadata server load

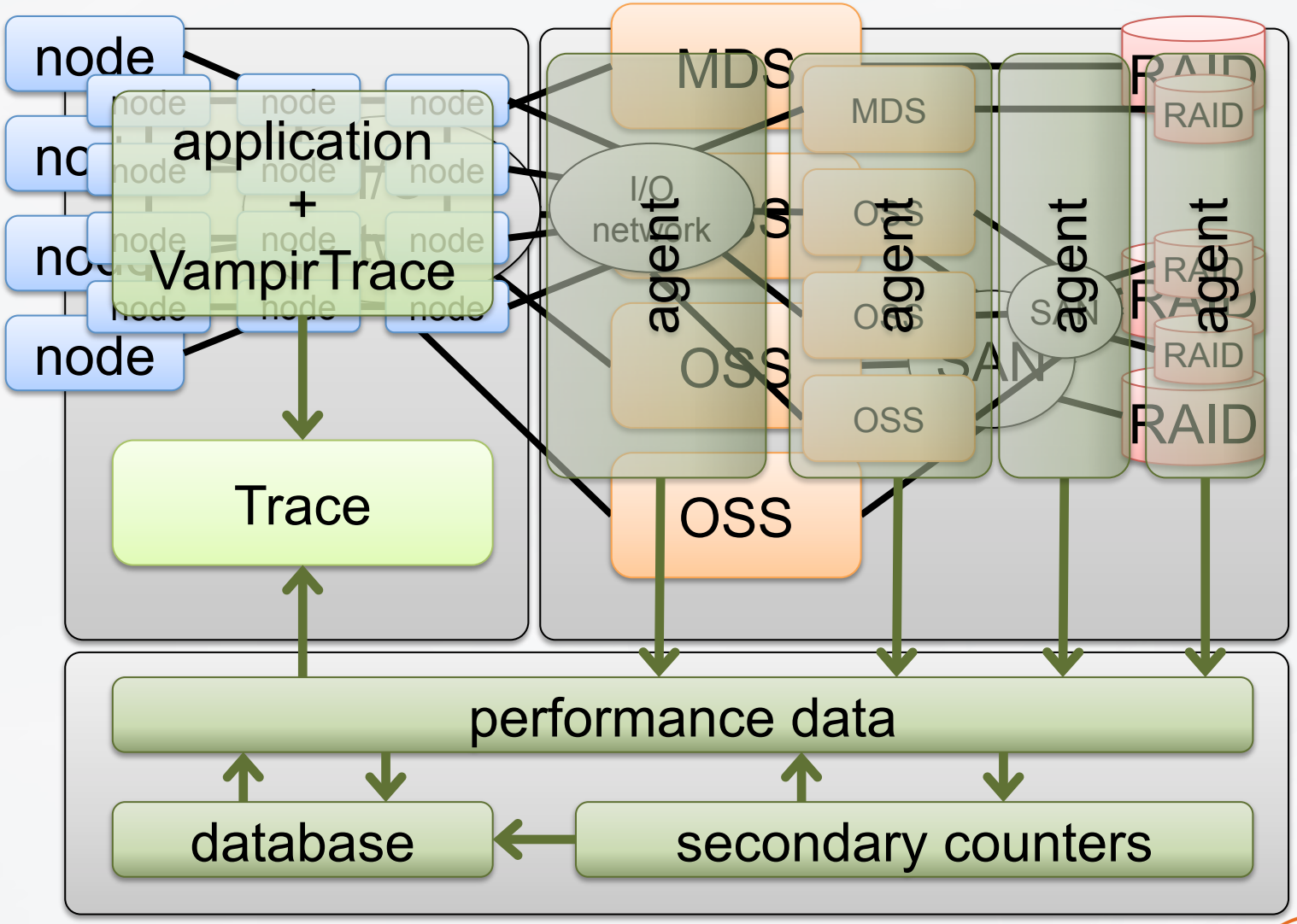
detailed per file and summary statistics

detailed per event data

Meeting the Challenges – Vampir I/O Analysis



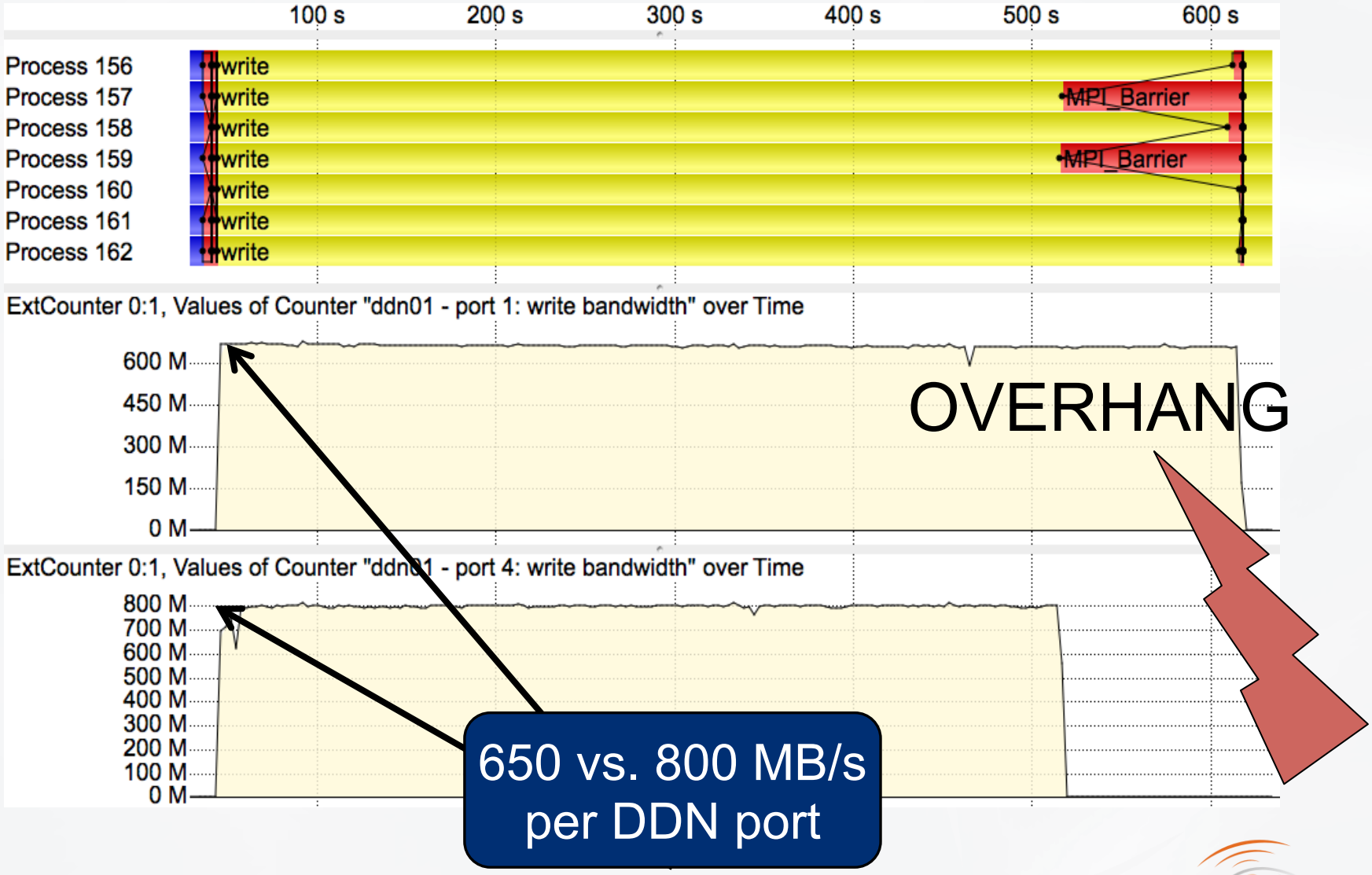
Meeting the Challenges – Vampir I/O Analysis



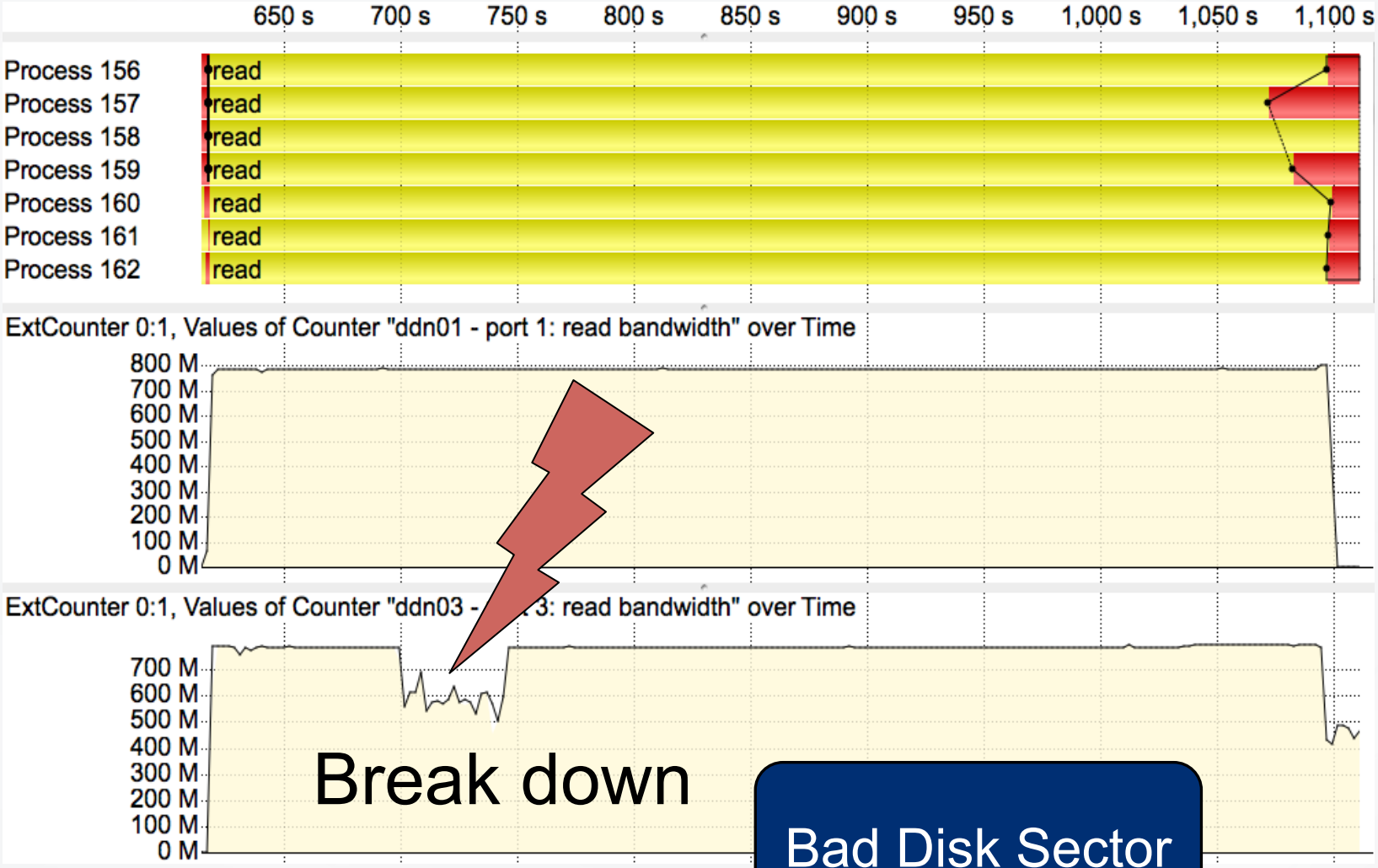
Starting Point: 6 from 576 IOR processes

IOR: write phase

IOR: Write Phase



IOR: Read Phase



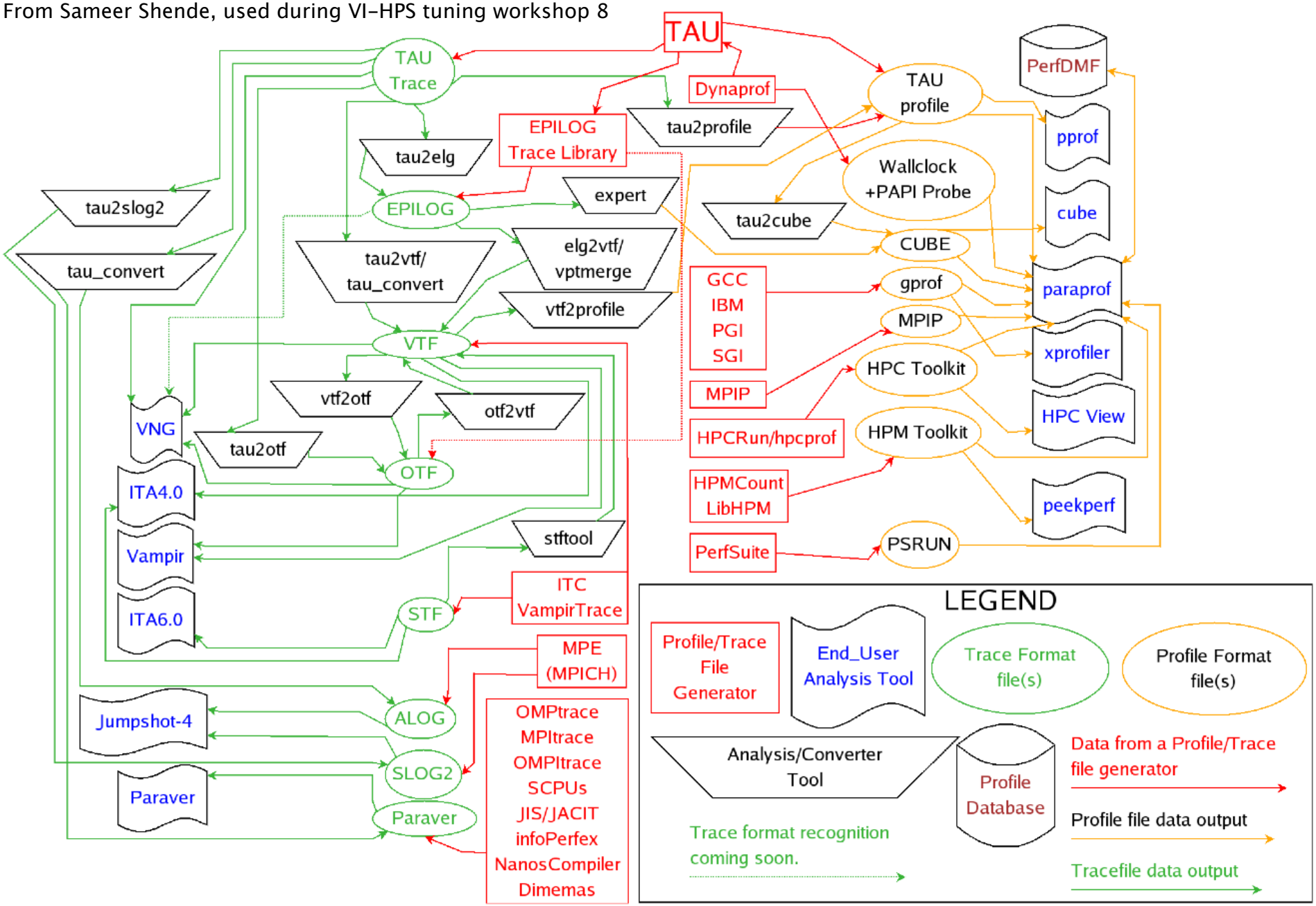
Bad Disk Sector

Content

- Overview
- Technology
- Example
- Tracing
- Visualization
- Meeting the Challenges
- Score-P
- Roadmap

Score-P – Motivation

From Sameer Shende, used during VI-HPS tuning workshop 8



Score-P – Overview

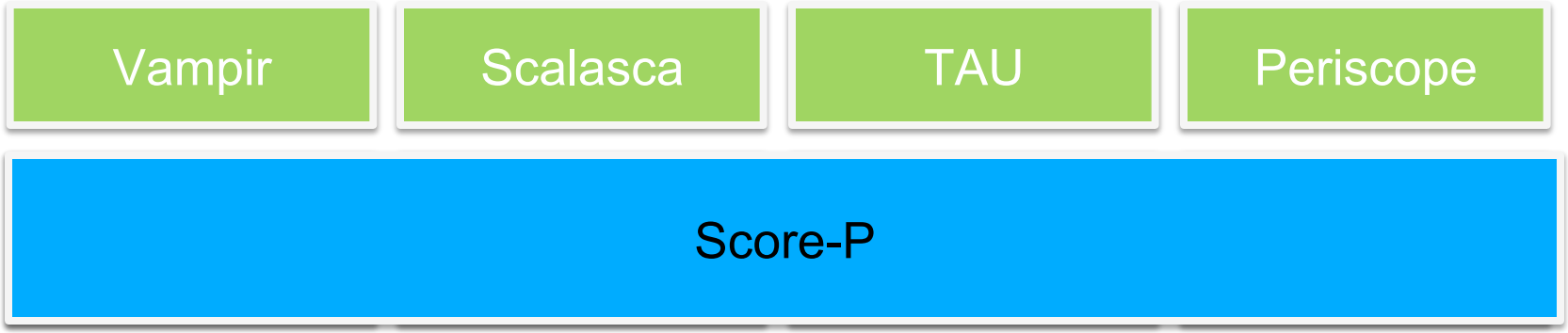
- Tool ecosystem:

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

- Consequence:
  - Each tool requires its own measurement system
  - Tool combinations limited
  - Higher tool development effort

Score-P – Overview

- Tool ecosystem:



- Goals:
  - Single measurement system
  - Better learning curve
  - More tool integration
  - Reduced tool development costs

Score-P – Partners

- Forschungszentrum Jülich, Germany
- German Research School for Simulation Sciences, Aachen, Germany
- Gesellschaft für numerische Simulation mbH Braunschweig, Germany
- RWTH Aachen, Germany
- Technische Universität Dresden, Germany
- Technische Universität München, Germany
- University of Oregon, Eugene, USA

Score-P – State

- Preview versions since 2010
- First release December 2011
- Score-P will replace VampirTrace
- Feature migrations are ongoing:
  - CUDA support on its way
  - TAU PDT support available
  - OPARI2 (in Score-P) supersedes OPARI in VT
  - …
- Vampir Tutorials will use Score-P starting mid 2012

Score-P – Usage

- Prefix commands provide instrumentation:

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

```
CC=scorep icc
CXX=scorep icpc
F90=scorep ifort
MPICC=scorep mpicc
```

- As with VampirTrace, environmentals control tool:
  - Profile/Tracing mode
  - Buffer sizes
  - Optional features

Content

- Overview
- Technology
- Example
- Tracing
- Visualization
- Meeting the Challenges
- Score-P
- Roadmap

Roadmap

June 2012

- Thinned out trace files (removal of similar/ redundant tasks)
- Temporal pattern highlighting during tracing
- OpenMP 3.0 task support
- VampirServer: improved remote connections
- ScoreP becomes official measurement system for Vampir
- CUDA 4.1 support with native CUPTI based tracing

November 2012

- Further MPI performance metrics in performance radar
- Support for HMPP Instrumentation API
- OTF2 extensions
- Temporal pattern based trace compression
- Score-P: Scalability, rewind, selective tracing

Longer term

- In memory performance event data compression
- Alignment based trace comparison
- Improved correlation of event types.
- Scalability improvements
- Score-P: measurement of system background counters (I/O, energy, …)
- Vampir: Background counter view
- PGAS/SHMEM support

Part III: Runtime Error Detection with MUST

Dr. Matthias S. Müller
Tobias Hilbrich

Email: {matthias.mueller, tobias.hilbrich}@tu-dresden.de

ZIH
Center for Information Services & High Performance Computing

Content

- Motivation
- Overview
- MPI Usage Errors
- Technology
- Example
- Usage
- Meeting the Challenges

Content

- Motivation
- Overview
- MPI Usage Errors
- Technology
- Example
- Usage
- Meeting the Challenges

Motivation

- MPI programming is error prone
- Results of an application run:
  - Crash
  - Application hanging
  - Result
- Questions:
  - Why crash/hang?
  - Is my result correct?
  - Will my code also give correct results on another system?

Content

- Motivation
- Overview
- MPI Usage Errors
- Technology
- Example
- Usage
- Meeting the Challenges

Overview

- Example:

Fortran type in C

```
...
MPI_Type_contiguous (2, MPI_INTEGER,
&newtype);
MPI_Send (buf, count, newtype, target,
          tag, MPI_COMM_WORLD)
...
```

Use of uncommited type

- MUST:
  - Detects MPI usage errors at runtime
  - A library that is attached to application
  - Strengths: Deadlock detection, collective verification, type matching
  - Goal: scalable correctness checks
  - Cooperation with LLNL and LANL

Content

- Motivation
- Overview
- MPI Usage Errors
- Technology
- Example
- Usage
- Meeting the Challenges

MPI Usage Errors – Incorrect Arguments

- Complications
  - Calls with many arguments
  - Fortran: many arguments have type INTEGER
  - Several argument restrictions
- Example:

Fortran datatype in C

```
MPI_Send(
    buf,
    count,
    MPI_INTEGER,
    target,
    tag,
    MPI_COMM_WORLD);
```

MPI Usage Errors – Resource Usage

- Complications
  - Many types of resources
  - Leaks
  - MPI internal limits
- Example:

```
MPI_Isend (..., &request);
MPI_Finalize ();
```

Applications should complete the outstanding communication associated with *request*

MPI Usage Errors – Buffer Overlaps

- Complications
  - Memory regions passed to MPI must not overlap (except send-send)
  - Derived datatypes can span non-contiguous regions
  - Collectives can both send and receive
- Example:

```
MPI_Isend (&(buf[0])/*buf*/, 5/*count*/, MPI_INT,...);
MPI_Irecv (&(buf[4])/*buf*/, 5/*count*/, MPI_INT,...);
```

Overlaps element buf[4] from the MPI_Isend call!

MPI Usage Errors – Type Matching

- Complications
  - Complex derived types
  - Types match if the signature matches, not their constructors
  - Partial receives
- Example 1:

Task 0

```
MPI_Send (buf, 1, MPI_INT)
```

Task 1

```
MPI_Recv (buf, 1, MPI_INT)
```

No Error, types match

MPI Usage Errors – Type Matching (2)

- Example 2:
  - Consider type T1 = {MPI_INT, MPI_INT}

Task 0

```
MPI_Send (buf, 1, T1)
```

Task 1

```
MPI_Recv (buf, 2, MPI_INT)
```

No Error, types match

- Example 3:
  - T1 = {MPI_INT, MPI_FLOAT}
  - T2 = {MPI_INT, MPI_INT}

Task 0

```
MPI_Send (buf, 1, T1)
```

Task 1

```
MPI_Recv (buf, 1, T2)
```

Missmatch: MPI_FLOAT != MPI_INT

MPI Usage Errors – Deadlocks

- Complications:
  - Non-blocking communication
  - Complex completions (Wait{all, any, some})
  - Non-determinism (e.g. MPI_ANY_SOURCE)
  - Choices of implementation in MPI standard
- Example:

Task 0

```
MPI_Recv (from:1)
```

Task 1

```
MPI_Recv (from:0)
```

Deadlock: tasks wait for each other

MPI Usage Errors – Deadlocks (2)

- How to visualise/understand deadlocks?
  - Common approach waiting-for graphs (WFGs)
  - One node for each rank
  - Rank X waits for rank Y => node X has arc to node Y
- Consider situation from Example 1:

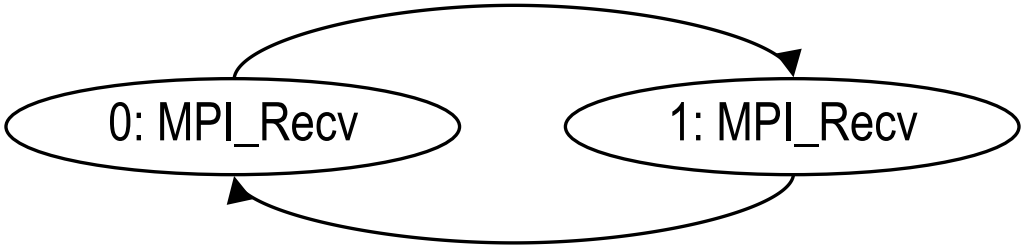
Task 0

```
MPI_Recv (from:1)
```

Task 1

```
MPI_Recv (from:0)
```

- Visualization:

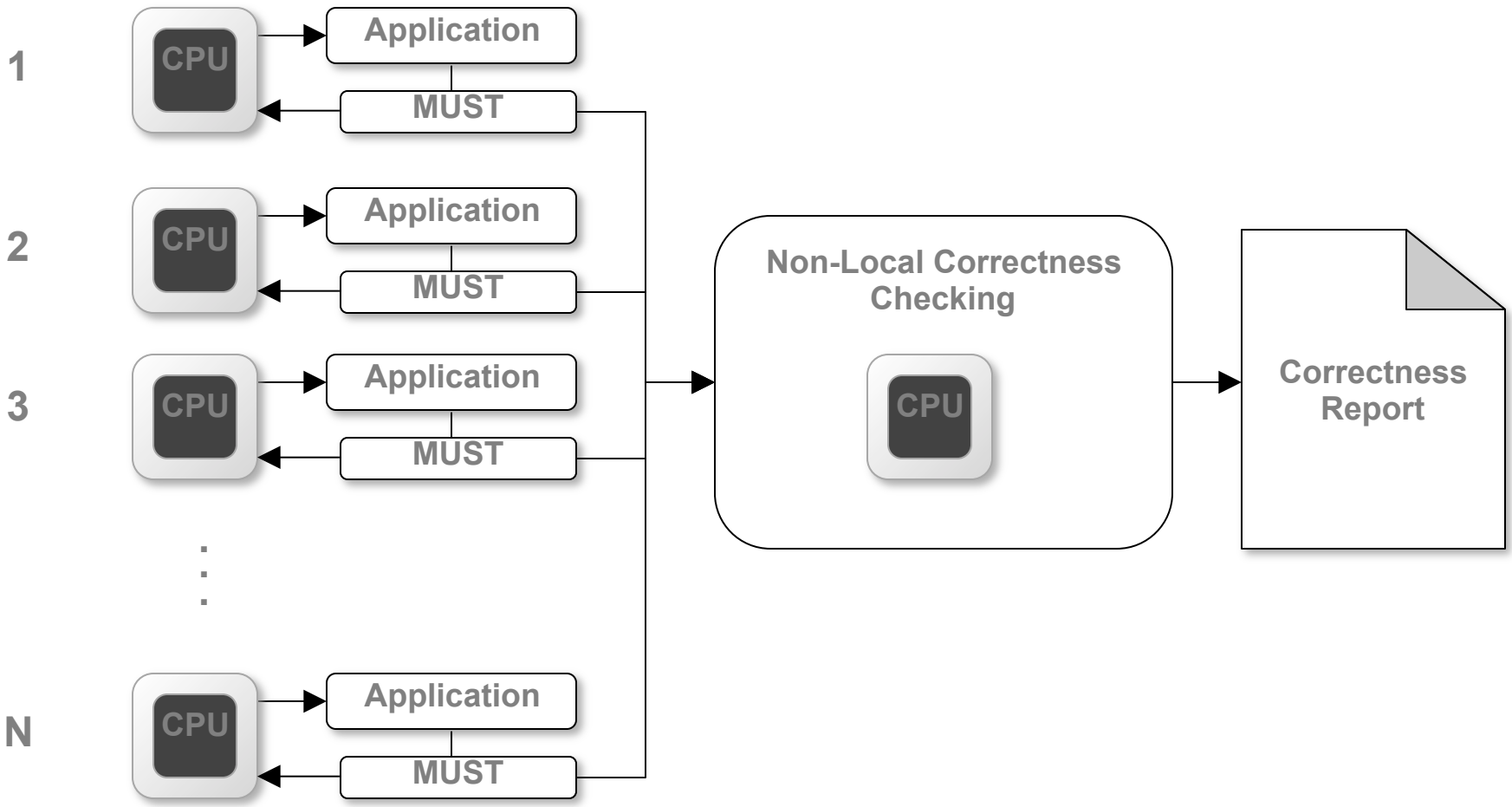


- Deadlock criterion: cycle (For simple cases)

Content

- Motivation
- Overview
- MPI Usage Errors
- Technology
- Example
- Usage
- Meeting the Challenges

Technology


1
CPU
Application
MUST
2
CPU
Application
MUST
3
CPU
Application
MUST
N
CPU
Application
MUST
Non-Local Correctness Checking
CPU
Correctness Report

Content

- Motivation
- Overview
- MPI Usage Errors
- Technology
- Example
- Usage
- Meeting the Challenges

Example – Synthetic Code

```
MPI_Init (&argc,&argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);

//1) Create a datatype
MPI_Type_contiguous (2, MPI_INT, &newType);
MPI_Type_commit (&newType);

//2) Use MPI_Sendrecv to perform a ring communication
MPI_Sendrecv (
        sBuf, 1, newType, (rank+1)%size, 123,
        rBuf, sizeof(int)*2, MPI_BYTE, (rank-1+size) % size, 123,
        MPI_COMM_WORLD, &status);

//3) Use MPI_Send and MPI_Recv to perform a ring communication
MPI_Send (    sBuf, 1, newType, (rank+1)%size, 456,
              MPI_COMM_WORLD);
MPI_Recv (    rBuf, sizeof(int)*2, MPI_BYTE, (rank-1+size) % size, 456,
              MPI_COMM_WORLD, &status);

MPI_Finalize ();
```

Example – Type mismatch

Rank	Thread	Type	Message	From	References	MPI-Standard Reference
1		**Error**	A send and a receive operation use datatypes that do not match! Missmatch occurs at (CONTIGUOUS)[0] (MPI_INT) in the send type and at (MPI_BYTE) in the receive type (consult the MUST manual for a detailed description of datatype positions). The send operation was started at reference 1, the receive operation was started at reference 2. (Information on communicator: MPI_COMM_WORLD) (Information on send of count 1 with type:Datatype created at reference 3 is for C, commited at reference 4, based on the following type(s): { MPI_INT}Typemap = {(MPI_INT, 0), (MPI_INT, 4)}) (Information on receive of count 8 with type:MPI_BYTE)	**MPI_Sendrecv** from: #0 main@test.c:54 #1 start_main@libc.so	reference 1 rank 0: **MPI_Sendrecv** from: #0 main@test.c:54 #1 start_main@libc.so reference 2 rank 1: **MPI_Sendrecv** from: #0 main@test.c:54 #1 start_main@libc.so reference 3 rank 0: **MPI_Type_contiguous** from: #0 main@test.c:35 #1 start_main@libc.so reference 4 rank 0: **MPI_Type_commit** from: #0 main@test.c:51 #1 start_main@libc.so	

Example – Type not freed

Rank	Thread	Type	Message	From	References	MPI-Standard Reference
		Error	There are 1 datatypes that are not freed when MPI_Finalize was issued, a quality application should free all MPI resources before calling MPI_Finalize. Listing information for these datatypes: -Datatype 1: Datatype created at reference 1 is for C, commited at reference 2, based on the following type(s): { MPI_INT}Typemap = {(MPI_INT, 0), (MPI_INT, 4)}		reference 1 rank 2: **MPI_Type_contiguous** from: #0 main@test.c:50 #1 start_main@libc.so reference 2 rank 2: **MPI_Type_commit** from: #0 main@test.c:51 #1 start_main@libc.so	
		Error	There are 1 datatypes that are not freed when MPI_Finalize was issued, a quality application should free all MPI resources before calling MPI_Finalize. Listing information for these datatypes: -Datatype 1: Datatype created at reference 1 is for C, commited at reference 2, based on the following type(s): { MPI_INT}Typemap = {(MPI_INT, 0), (MPI_INT, 4)}		reference 1 rank 1: **MPI_Type_contiguous** from: #0 main@test.c:50 #1 start_main@libc.so reference 2 rank 1: **MPI_Type_commit** from: #0 main@test.c:51 #1 start_main@libc.so	

Example – Deadlock (potential)

MUST Outputfile

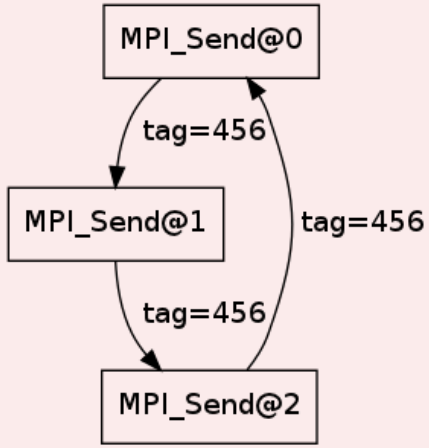
MUST Deadlock Details, date: Thu Mar 22 15:24:15 2012.

Back to MUST error report

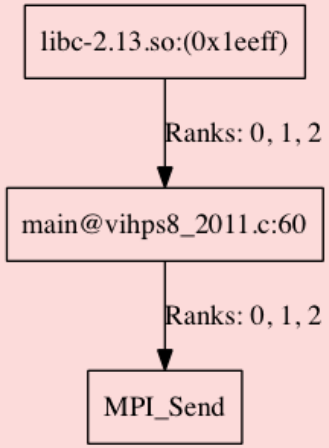
Message

The application issued a set of MPI calls that can cause a deadlock! The graphs below show details on this situation. This includes a wait-for graph that shows active waiting dependencies in the deadlock situation , a legend for this graph and a call stack view. The application still runs, if the deadlock manifested (e.g. caused a hang on this MPI implementation) you can attach to the involved ranks with a debugger or abort the application (if necessary).

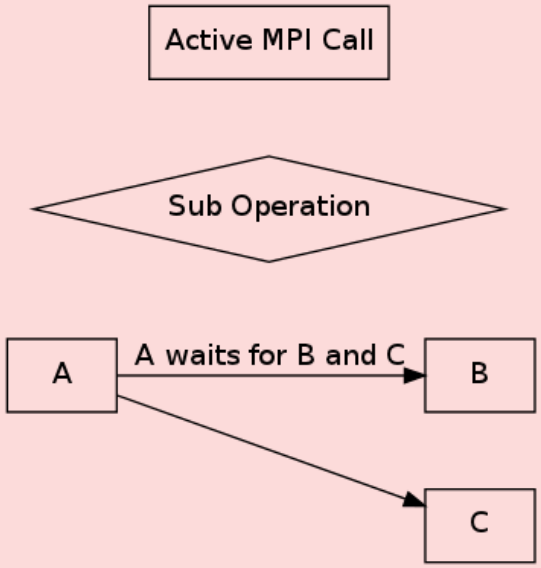
Wait-for Graph



Call Stack



Legend



Content

- Motivation
- Overview
- MPI Usage Errors
- Technology
- Example
- Usage
- Meeting the Challenges

Usage

- MUST uses an mpirun wrapper:

```
% mpicc source.c -o exe
% mpirun -np 4 ./exe
```

```
% mpicc source.c -o exe
% mustrun -np 4 ./exe
```

- After run: inspect “MUST_Output.html”
- “mustrun” uses an extra process:
  - I.e.: “mustrun –np 4 …” will use 5 tasks
  - Allocate the extra resource in batch jobs

Usage – Operation Modes

- MUST causes overhead at runtime
- MUST expects a crash at any time
- Blocking communication ensures error detection

⇒ Can cause high overhead

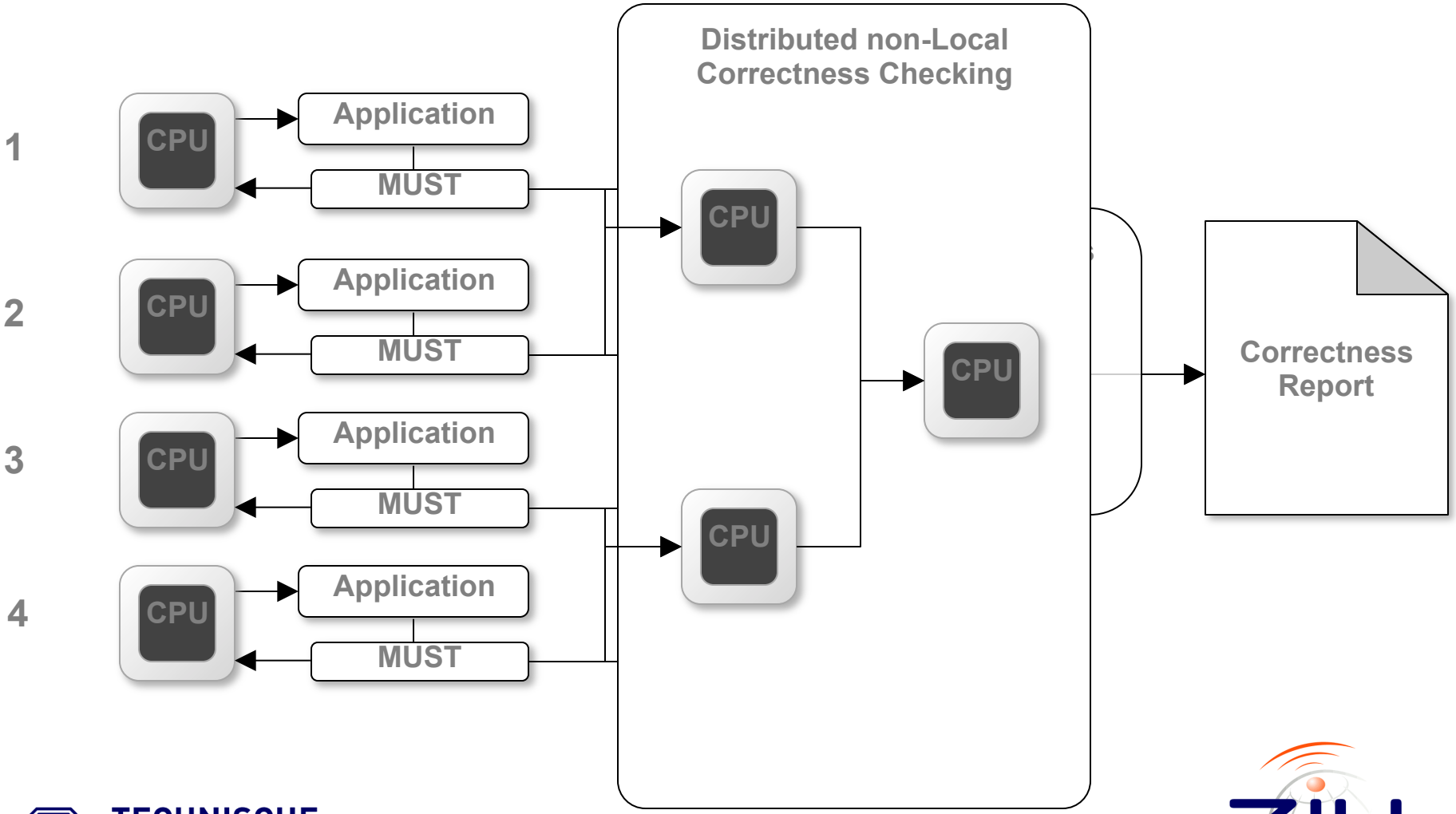
- If application does not crash:
  - Add “--must:nocrash” to the mustrun command
  - Uses aggregated non-blocking communication
  - Provides substantial speed up
- More options: “mustrun –help”

Content

- Motivation
- Overview
- MPI Usage Errors
- Technology
- Example
- Usage
- Meeting the Challenges

Meeting the Challenges

- Currently: core concern scalability



Meeting the Challenges – Scalability

- Local checks (Scalable):
  - Integer validation
  - Integrity checks (pointers valid, etc.)
  - Operation, Request, Communicator, Datatype, Group usage
  - Resource leak detection
  - Memory overlap checks
- Non-local checks (Scalability in progress):
  - Collective verification
  - Lost message detection
  - Type matching (For P2P and collectives)
  - Deadlock detection (with root cause visualization)

Part 4 VI: Conclussions & Wrap Up

Dr. Matthias S. Müller
Tobias Hilbrich

Email: {matthias.mueller, tobias.hilbrich}@tu-dresden.de

ZIH
Center for Information Services & High Performance Computing

Tool Availability

- VampirTrace: Open source
  - ⇒ http://tinyurl.com/vampirtrace
- Vampir: Commercial; LLNL/LANL have licenses
  - ⇒ http://www.vampir.eu/
- MUST: Open source
  - ⇒ http://tu-dresden.de/zih/must/
- VM with full installation available, contact us
- Contact:
  - {tobias.hilbrich, matthias.mueller}@tu-dresden.de
  - Bugs/whishes/ideas:
    vampirsupport@zih.tu-dresden.de

Conclusions

- Vampir suite:
  - Tool for performance optimization
  - Many recent extensions
  - Scalable to 200.000 cores
  - New features include:
    - CUDA support, Counter overlays, Clustering, PDT/Dyninst instrumentation, Search functions, Performance radar, derived counters, Trace Rewind
- MUST:
  - Tool to detect MPI usage errors at runtime
  - Scalable to about 1000 cores (more with less checks)
  - 10.000 core scalability in development