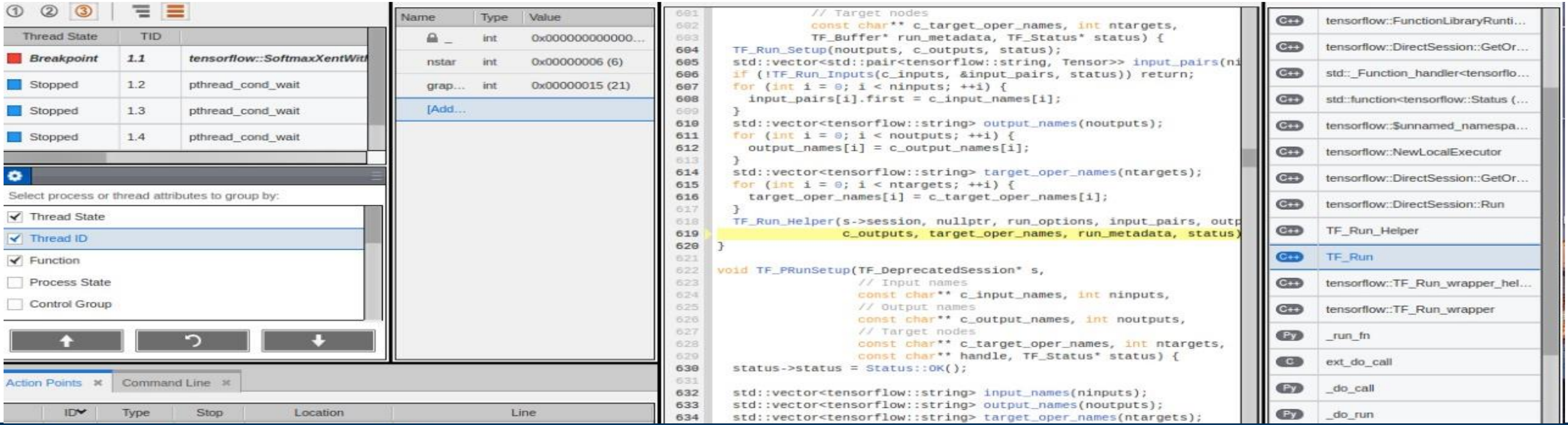
MPI/OPENMP COURSE – TOOLS

MARCH 20, 2024 I MICHAEL KNOBLOCH I M.KNOBLOCH@FZ-JUELICH.DE

Mitglied der Helmholtz-Gemeinschaft

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE



Thread State	TID	
🟥 ***Breakpoint***	***1.1***	***tensorflow::SoftmaxXentWitł***
🟦 Stopped	1.2	pthread_cond_wait
🟦 Stopped	1.3	pthread_cond_wait
🟦 Stopped	1.4	pthread_cond_wait

☑ Thread State
☑ Thread ID
☑ Function
☐ Process State
☐ Control Group

Name	Type	Value
_	int	0x000000000000…
nstar	int	0x00000006 (6)
grap…	int	0x00000015 (21)
[Add…		

```
               // Target nodes
               const char** c_target_oper_names, int ntargets,
               TF_Buffer* run_metadata, TF_Status* status) {
  TF_Run_Setup(noutputs, c_outputs, status);
  std::vector<std::pair<tensorflow::string, Tensor>> input_pairs(ni
  if (!TF_Run_Inputs(c_inputs, &input_pairs, status)) return;
  for (int i = 0; i < ninputs; ++i) {
    input_pairs[i].first = c_input_names[i];
  }
  std::vector<tensorflow::string> output_names(noutputs);
  for (int i = 0; i < noutputs; ++i) {
    output_names[i] = c_output_names[i];
  }
  std::vector<tensorflow::string> target_oper_names(ntargets);
  for (int i = 0; i < ntargets; ++i) {
    target_oper_names[i] = c_target_oper_names[i];
  }
  TF_Run_Helper(s->session, nullptr, run_options, input_pairs, outp
                c_outputs, target_oper_names, run_metadata, status)
}

void TF_PRunSetup(TF_DeprecatedSession* s,
                  // Input names
                  const char** c_input_names, int ninputs,
                  // Output names
                  const char** c_output_names, int noutputs,
                  // Target nodes
                  const char** c_target_oper_names, int ntargets,
                  const char** handle, TF_Status* status) {
  status->status = Status::OK();

  std::vector<tensorflow::string> input_names(ninputs);
  std::vector<tensorflow::string> output_names(noutputs);
  std::vector<tensorflow::string> target_oper_names(ntargets);
```

C++	tensorflow::FunctionLibraryRunti…
C++	tensorflow::DirectSession::GetOr…
C++	std::_Function_handler<tensorflo…
C++	std::function<tensorflow::Status (…
C++	tensorflow::$unnamed_namespa…
C++	tensorflow::NewLocalExecutor
C++	tensorflow::DirectSession::GetOr…
C++	tensorflow::DirectSession::Run
C++	TF_Run_Helper
C++	TF_Run
C++	tensorflow::TF_Run_wrapper_hel…
C++	tensorflow::TF_Run_wrapper
Py	_run_fn
C	ext_do_call
Py	_do_call
Py	_do_run

MUST – MPI CORRECTNESS CHECKER

HOW MANY ISSUES CAN YOU SPOT?

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int rank, size, buf[8];

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Datatype type;
    MPI_Type_contiguous(2, MPI_INTEGER, &type);

    MPI_Recv(buf, 2, MPI_INT, size - rank, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    MPI_Send (buf, 2, type, size - rank, 123, MPI_COMM_WORLD);

    printf ("Hello, I am rank %d of %d.\n", rank, size);

    return 0;
}
```

At least 8 issues in this example!

MOTIVATION

- MPI programming is error prone
- Portabillity errors
  (just on some systems, runs, configurations)
- Bugs may manifest as
  - Crash
  - Application hanging
  - Application finishes
- Questions
  - Why crash/hang?
  - Is the result correct?
  - Will the code produce the correct result on another system?

Error more obvious

- Tools help to pin-point these issues

TYPES OF ERRORS

- Common syntactic errors:
  - Incorrect arguments
  - Resource usage
  - Lost/Dropped Requests
  - Buffer usage
  - Type-matching
  - Deadlocks

Tool to use:

MUST,

Static analysis tool,

(Debugger)

- Semantic errors that are correct in terms of MPI standard, but do not match the programmer's intention:
  - Displacement/Size/Count errors

Tool to use:

Debugger

MUST

- Next generation MPI correctness and portability checker
- https://www.i12.rwth-aachen.de/go/id/nrbe

- MUST reports
  - Errors: violations of the MPI-standard
  - Warnings: unusual behavior or possible problems
  - Notes: harmless but remarkable behavior
  - Potential deadlock detection

- Usage
  - Compile with debug information (i.e. use the -g flag)
  - Run application under the control of mustrun (requires (at least) one additional MPI process)
    - E.g. on JUSUF: mustrun --must:mpiexec srun --must:np -n -n 4 ./app
  - Open output html report (might need to copy it to your local machine)

FIX 0: ADD MPI_INIT/MPI_FINALIZE

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int rank, size, buf[8];

   MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Datatype type;
    MPI_Type_contiguous(2, MPI_INTEGER, &type);

    MPI_Recv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    MPI_Send (buf, 2, type, size - rank - 1, 123, MPI_COMM_WORLD);

   printf ("Hello, I am rank %d of %d.\n", rank, size);

   MPI_Finalize();

   return 0;
}
```

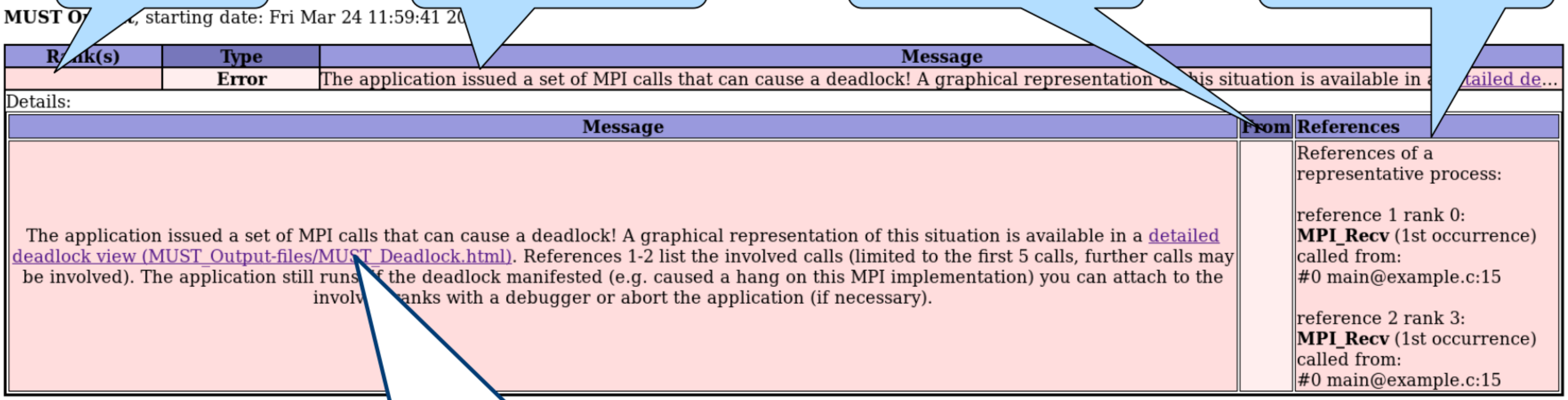
MUST DETECTS DEADLOCKS

Who?

What?

Where?

Details



MUST Output, starting date: Fri Mar 24 11:59:41 20

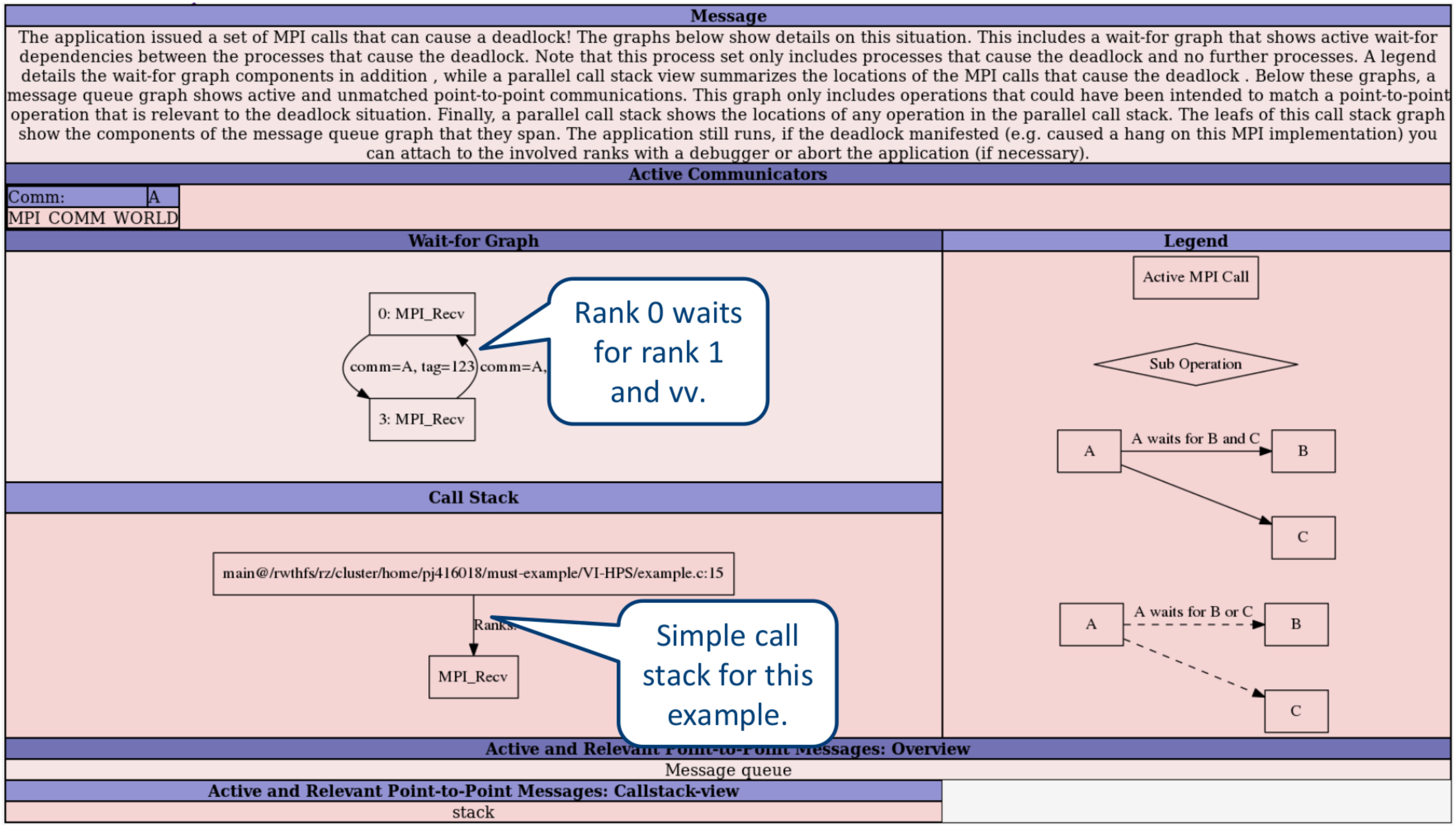
Rank(s)	Type	Message
	Error	The application issued a set of MPI calls that can cause a deadlock! A graphical representation of this situation is available in a detailed de...

Details:

Message	From	References
The application issued a set of MPI calls that can cause a deadlock! A graphical representation of this situation is available in a detailed deadlock view (MUST_Output-files/MUST_Deadlock.html). References 1-2 list the involved calls (limited to the first 5 calls, further calls may be involved). 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).		References of a representative process: reference 1 rank 0: **MPI_Recv** (1st occurrence) called from: #0 main@example.c:15 reference 2 rank 3: **MPI_Recv** (1st occurrence) called from: #0 main@example.c:15

Click for graphical representation of the detected deadlock situation.

MUST DETECTS DEADLOCKS



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 wait-for dependencies between the processes that cause the deadlock. Note that this process set only includes processes that cause the deadlock and no further processes. A legend details the wait-for graph components in addition , while a parallel call stack view summarizes the locations of the MPI calls that cause the deadlock . Below these graphs, a message queue graph shows active and unmatched point-to-point communications. This graph only includes operations that could have been intended to match a point-to-point operation that is relevant to the deadlock situation. Finally, a parallel call stack shows the locations of any operation in the parallel call stack. The leafs of this call stack graph show the components of the message queue graph that they span. 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).

Active Communicators

Comm:	A
MPI_COMM_WORLD	

Wait-for Graph

Rank 0 waits for rank 1 and vv.

Legend

Call Stack

main@/rwthfs/rz/cluster/home/pj416018/must-example/VI-HPS/example.c:15
Ranks:
MPI_Recv

Simple call stack for this example.

Active and Relevant Point-to-Point Messages: Overview

Message queue

Active and Relevant Point-to-Point Messages: Callstack-view

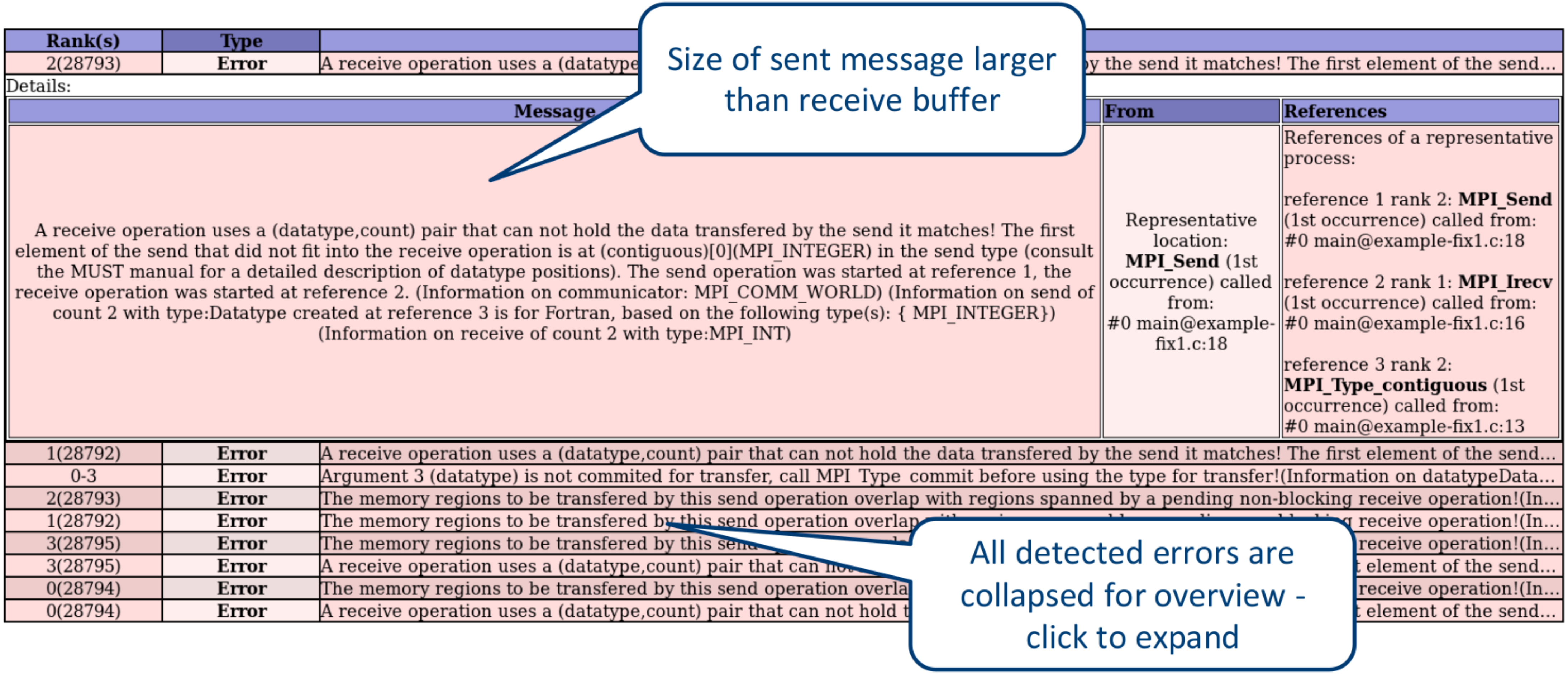
stack

FIX 1: USE ASYNC RECV

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int rank, size, buf[8];

   MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Datatype type;
    MPI_Type_contiguous(2, MPI_INTEGER, &type);

    MPI_Request request;
    MPI_Irecv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, &request);

    MPI_Send (buf, 2, type, size - rank - 1, 123, MPI_COMM_WORLD);

   printf ("Hello, I am rank %d of %d.\n", rank, size);

   MPI_Finalize();

   return 0;
}
```

Use asynchronous receive (MPI_Irecv)

MUST DETECTS BUFFER ERRORS



Rank(s)	Type	
2(28793)	Error	A receive operation uses a (datatype y the send it matches! The first element of the send...

Details:

Message	From	References
A receive operation uses a (datatype,count) pair that can not hold the data transfered by the send it matches! The first element of the send that did not fit into the receive operation is at (contiguous)[0](MPI_INTEGER) in the send 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 2 with type:Datatype created at reference 3 is for Fortran, based on the following type(s): { MPI_INTEGER}) (Information on receive of count 2 with type:MPI_INT)	Representative location: **MPI_Send** (1st occurrence) called from: #0 main@example-fix1.c:18	References of a representative process: reference 1 rank 2: **MPI_Send** (1st occurrence) called from: #0 main@example-fix1.c:18 reference 2 rank 1: **MPI_Irecv** (1st occurrence) called from: #0 main@example-fix1.c:16 reference 3 rank 2: **MPI_Type_contiguous** (1st occurrence) called from: #0 main@example-fix1.c:13

Rank(s)	Type	
1(28792)	Error	A receive operation uses a (datatype,count) pair that can not hold the data transfered by the send it matches! The first element of the send...
0-3	Error	Argument 3 (datatype) is not commited for transfer, call MPI_Type_commit before using the type for transfer!(Information on datatypeData...
2(28793)	Error	The memory regions to be transfered by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...
1(28792)	Error	The memory regions to be transfered by this send operation overlap g receive operation!(In...
3(28795)	Error	The memory regions to be transfered by this sen receive operation!(In...
3(28795)	Error	A receive operation uses a (datatype,count) pair that can no t element of the send...
0(28794)	Error	The memory regions to be transfered by this send operation overla receive operation!(In...
0(28794)	Error	A receive operation uses a (datatype,count) pair that can not hold t t element of the send...

Size of sent message larger than receive buffer

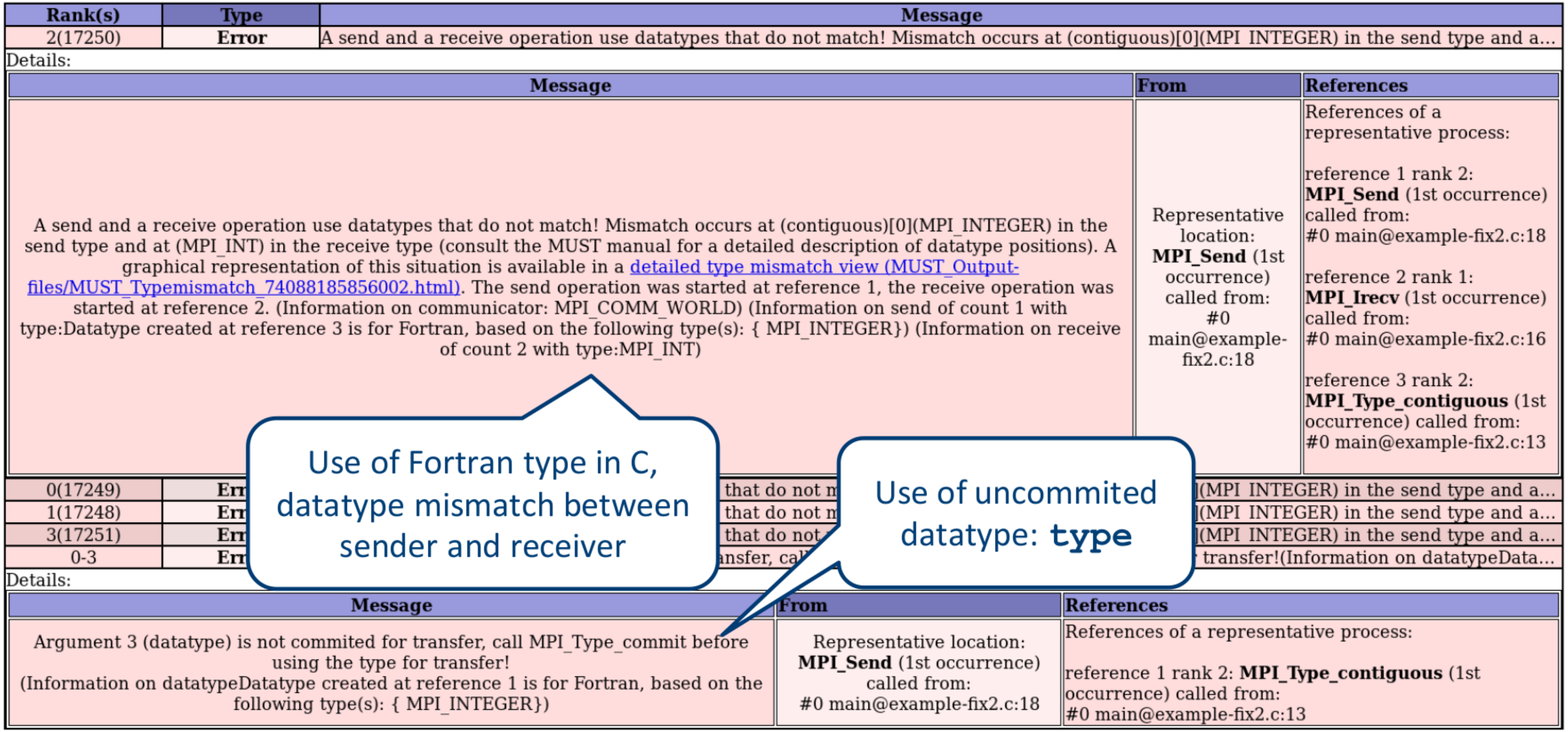
All detected errors are collapsed for overview - click to expand

FIX 2: SAME MESSAGE SIZE FOR SEND/RECV

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int rank, size, buf[8];

   MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Datatype type;
    MPI_Type_contiguous(2, MPI_INTEGER, &type);

    MPI_Request request;
    MPI_Irecv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, &request);

    MPI_Send (buf, 1, type, size - rank - 1, 123, MPI_COMM_WORLD);

    printf ("Hello, I am rank %d of %d.\n", rank, size);

    MPI_Finalize();

    return 0;
}
```

Reduce the message size

MUST DETECTS DATATYPE ERRORS



Rank(s)	Type	Message
2(17250)	Error	A send and a receive operation use datatypes that do not match! Mismatch occurs at (contiguous)[0](MPI INTEGER) in the send type and a...

Details:

Message	From	References
A send and a receive operation use datatypes that do not match! Mismatch occurs at (contiguous)[0](MPI_INTEGER) in the send type and at (MPI_INT) in the receive type (consult the MUST manual for a detailed description of datatype positions). A graphical representation of this situation is available in a [detailed type mismatch view (MUST_Output-files/MUST_Typemismatch_74088185856002.html)](MUST_Output-files/MUST_Typemismatch_74088185856002.html). 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 Fortran, based on the following type(s): { MPI_INTEGER}) (Information on receive of count 2 with type:MPI_INT)	Representative location: **MPI_Send** (1st occurrence) called from: #0 main@example-fix2.c:18	References of a representative process: reference 1 rank 2: **MPI_Send** (1st occurrence) called from: #0 main@example-fix2.c:18 reference 2 rank 1: **MPI_Irecv** (1st occurrence) called from: #0 main@example-fix2.c:16 reference 3 rank 2: **MPI_Type_contiguous** (1st occurrence) called from: #0 main@example-fix2.c:13

Rank(s)	Type	Message
0(17249)	Err	that do not m](MPI INTEGER) in the send type and a...
1(17248)	Err	that do not m](MPI INTEGER) in the send type and a...
3(17251)	Err	that do not](MPI INTEGER) in the send type and a...
0-3	Err	ansfer, cal transfer!(Information on datatypeData...

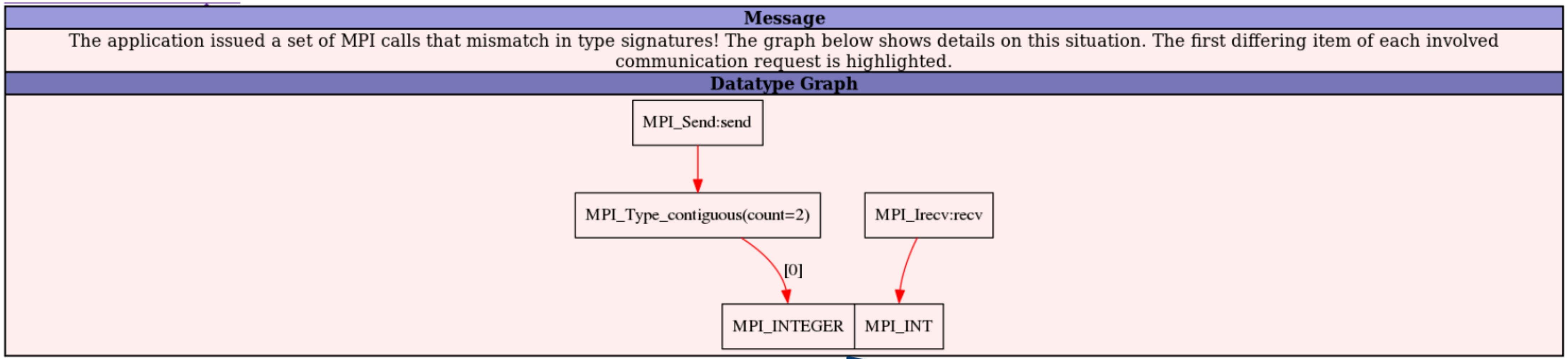
Details:

Message	From	References
Argument 3 (datatype) is not commited for transfer, call MPI_Type_commit before using the type for transfer! (Information on datatypeDatatype created at reference 1 is for Fortran, based on the following type(s): { MPI_INTEGER})	Representative location: **MPI_Send** (1st occurrence) called from: #0 main@example-fix2.c:18	References of a representative process: reference 1 rank 2: **MPI_Type_contiguous** (1st occurrence) called from: #0 main@example-fix2.c:13

Use of Fortran type in C, datatype mismatch between sender and receiver

Use of uncommited datatype: `type`

MUST DETECTS DATATYPE ERRORS



Message

The application issued a set of MPI calls that mismatch in type signatures! The graph below shows details on this situation. The first differing item of each involved communication request is highlighted.

Datatype Graph

Graphical representation of the type mismatch

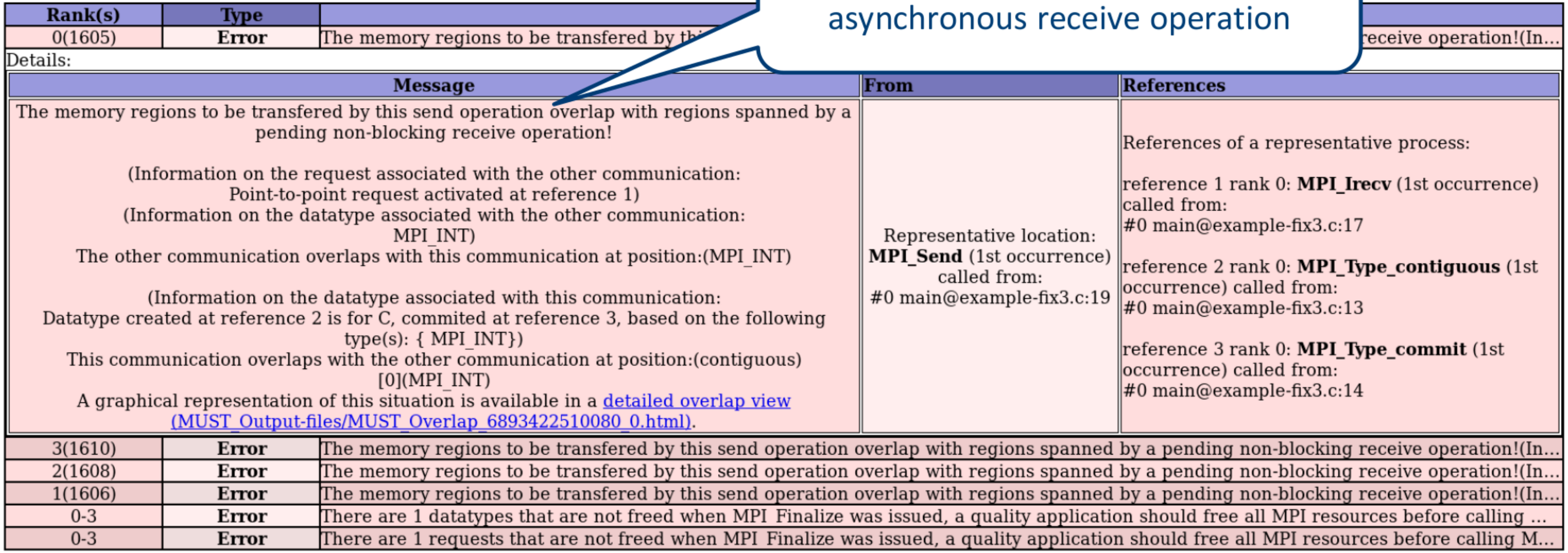
FIX 3+4: C INT TYPE & USE TYPE_COMMIT

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
   int rank, size, buf[8];

  MPI_Init(&argc, &argv);

   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   MPI_Datatype type;
   MPI_Type_contiguous(2, MPI_INT, &type);
   MPI_Type_commit(&type);

   MPI_Request request;
   MPI_Irecv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, &request);

   MPI_Send (buf, 1, type, size - rank - 1, 123, MPI_COMM_WORLD);

   printf ("Hello, I am rank %d of %d.\n", rank, size);

   MPI_Finalize();

  return 0;
}
```

Use integer datatype intended for C

Commit datatype before usage

MUST DETECTS DATARACES IN ASYNC COMM



Data race between send and asynchronous receive operation

Rank(s)	Type	
0(1605)	Error	The memory regions to be transfered by thi receive operation!(In...

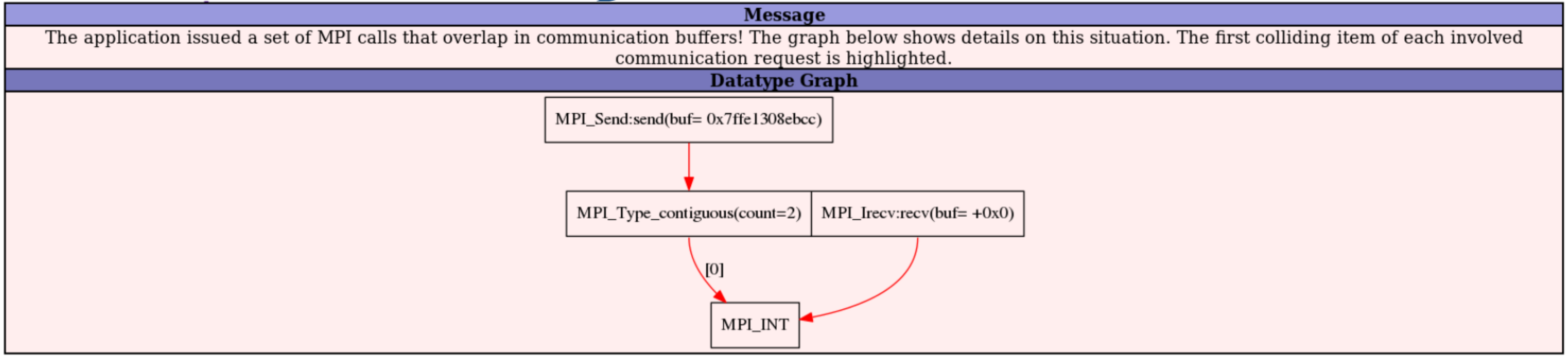
Details:

Message	From	References
The memory regions to be transfered by this send operation overlap with regions spanned by a pending non-blocking receive operation! (Information on the request associated with the other communication: Point-to-point request activated at reference 1) (Information on the datatype associated with the other communication: MPI_INT) The other communication overlaps with this communication at position:(MPI_INT) (Information on the datatype associated with this communication: Datatype created at reference 2 is for C, commited at reference 3, based on the following type(s): { MPI_INT}) This communication overlaps with the other communication at position:(contiguous)[0](MPI_INT) A graphical representation of this situation is available in a detailed overlap view (MUST_Output-files/MUST_Overlap_6893422510080_0.html).	Representative location: **MPI_Send** (1st occurrence) called from: #0 main@example-fix3.c:19	References of a representative process: reference 1 rank 0: **MPI_Irecv** (1st occurrence) called from: #0 main@example-fix3.c:17 reference 2 rank 0: **MPI_Type_contiguous** (1st occurrence) called from: #0 main@example-fix3.c:13 reference 3 rank 0: **MPI_Type_commit** (1st occurrence) called from: #0 main@example-fix3.c:14

3(1610)	Error	The memory regions to be transfered by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...
2(1608)	Error	The memory regions to be transfered by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...
1(1606)	Error	The memory regions to be transfered by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...
0-3	Error	There are 1 datatypes that are not freed when MPI_Finalize was issued, a quality application should free all MPI resources before calling ...
0-3	Error	There are 1 requests that are not freed when MPI_Finalize was issued, a quality application should free all MPI resources before calling M...

MUST DETECTS DATARACES IN ASYNC COMM

Graphical representation of the data race location



Message

The application issued a set of MPI calls that overlap in communication buffers! The graph below shows details on this situation. The first colliding item of each involved communication request is highlighted.

Datatype Graph

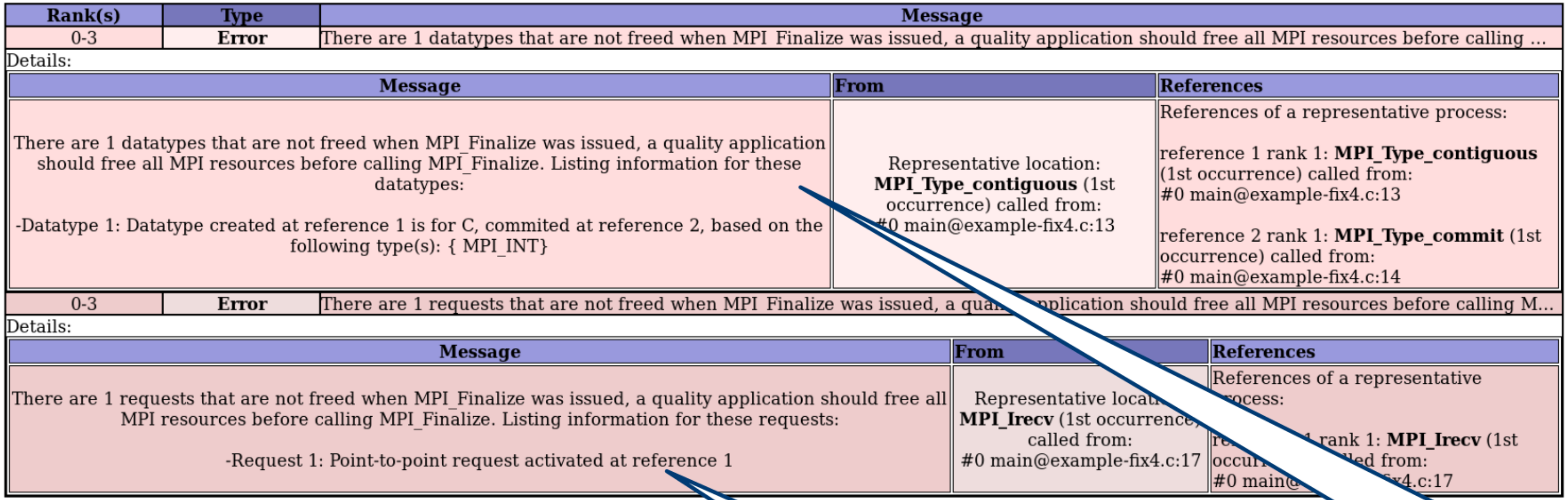
MPI_Send:send(buf= 0x7ffe1308ebcc)
MPI_Type_contiguous(count=2)
MPI_Irecv:recv(buf= +0x0)
[0]
MPI_INT

FIX 5: USE INDEPENDENT MEMORY REGIONS

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int rank, size, buf[8];

   MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Datatype type;
    MPI_Type_contiguous(2, MPI_INT, &type);
    MPI_Type_commit(&type);

    MPI_Request request;
    MPI_Irecv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, &request);

    MPI_Send (buf + 2, 1, type, size - rank - 1, 123, MPI_COMM_WORLD);

    printf ("Hello, I am rank %d of %d.\n", rank, size);

    MPI_Finalize();

    return 0;
}
```

Offset points to independent memory

MUST DETECTS LEAKS OF USER-DEFINED OBJECTS



Rank(s)	Type	Message
0-3	Error	There are 1 datatypes that are not freed when MPI_Finalize was issued, a quality application should free all MPI resources before calling ...

Details:

Message	From	References
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}	Representative location: **MPI_Type_contiguous** (1st occurrence) called from: #0 main@example-fix4.c:13	References of a representative process: reference 1 rank 1: **MPI_Type_contiguous** (1st occurrence) called from: #0 main@example-fix4.c:13 reference 2 rank 1: **MPI_Type_commit** (1st occurrence) called from: #0 main@example-fix4.c:14

Rank(s)	Type	Message
0-3	Error	There are 1 requests that are not freed when MPI_Finalize was issued, a quality application should free all MPI resources before calling M...

Details:

Message	From	References
There are 1 requests 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 requests: -Request 1: Point-to-point request activated at reference 1	Representative location: **MPI_Irecv** (1st occurrence) called from: #0 main@example-fix4.c:17	References of a representative process: reference 1 rank 1: **MPI_Irecv** (1st occurrence) called from: #0 main@example-fix4.c:17

Unfinished non-blocking receive is resource leak and missing synchronization

Leak of user defined datatype object

- User defined objects include
  - MPI_Comms (even by MPI_Comm_dup)
  - MPI_Datatypes
  - MPI_Groups

FIX 6+7: USE MPI_WAIT & FREE DATATYPE

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int rank, size, buf[8];

   MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Datatype type;
    MPI_Type_contiguous(2, MPI_INT, &type);
    MPI_Type_commit(&type);

    MPI_Request request;
    MPI_Irecv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, &request);

    MPI_Send (buf + 2, 1, type, size - rank - 1, 123, MPI_COMM_WORLD);
    MPI_Wait(&request, MPI_STATUS_IGNORE);

    printf ("Hello, I am rank %d of %d.\n", rank, size);
    MPI_Type_free(&type);

    MPI_Finalize();

    return 0;
}
```

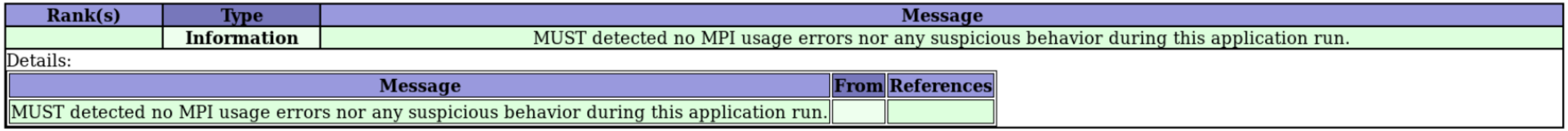
Finish asynchronous communication

Deallocate datatype

JÜLICH Forschungszentrum

JÜLICH SUPERCOMPUTING CENTRE

FINALLY



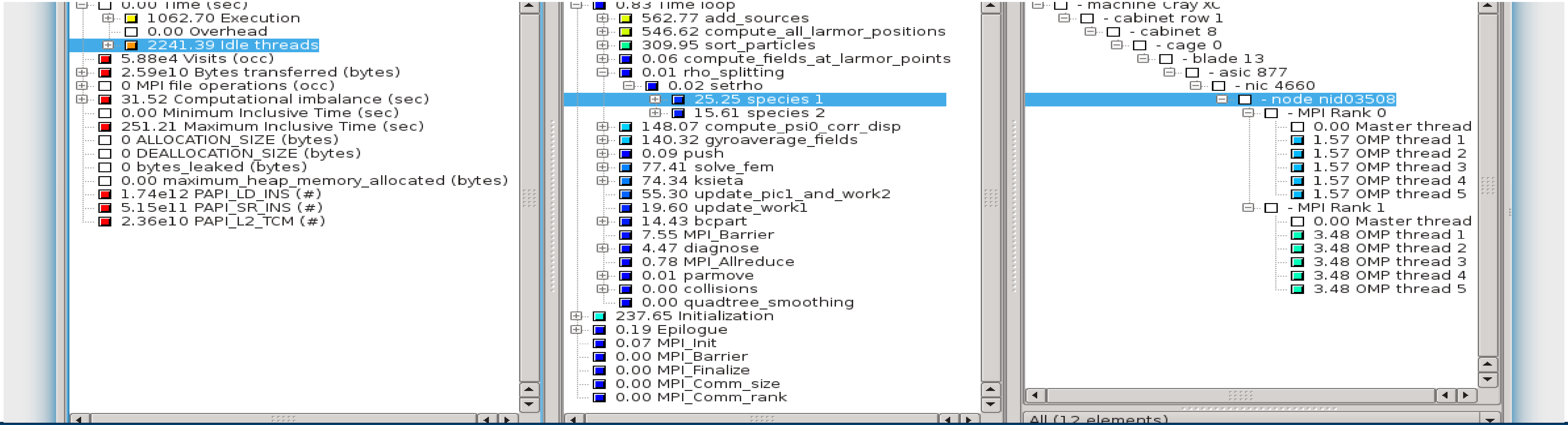
Rank(s)	Type	Message
	Information	MUST detected no MPI usage errors nor any suspicious behavior during this application run.

Details:

Message	From	References
MUST detected no MPI usage errors nor any suspicious behavior during this application run.		

No further error detected

Hopefully this message applies to many applications



```
0.00 Time (sec)
    1062.70 Execution
    0.00 Overhead
    2241.39 Idle threads
5.88e4 Visits (occ)
2.59e10 Bytes transferred (bytes)
0 MPI file operations (occ)
31.52 Computational imbalance (sec)
0.00 Minimum Inclusive Time (sec)
251.21 Maximum Inclusive Time (sec)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
1.74e12 PAPI_LD_INS (#)
5.15e11 PAPI_SR_INS (#)
2.36e10 PAPI_L2_TCM (#)
```

```
0.83 Time loop
    562.77 add_sources
    546.62 compute_all_larmor_positions
    309.95 sort_particles
    0.06 compute_fields_at_larmor_points
    0.01 rho_splitting
        0.02 setrho
            25.25 species 1
            15.61 species 2
    148.07 compute_psi0_corr_disp
    140.32 gyroaverage_fields
    0.09 push
    77.41 solve_fem
    74.34 ksieta
    55.30 update_pic1_and_work2
    19.60 update_work1
    14.43 bcpart
    7.55 MPI_Barrier
    4.47 diagnose
    0.78 MPI_Allreduce
    0.01 parmove
    0.00 collisions
    0.00 quadtree_smoothing
237.65 Initialization
0.19 Epilogue
0.07 MPI_Init
0.00 MPI_Barrier
0.00 MPI_Finalize
0.00 MPI_Comm_size
0.00 MPI_Comm_rank
```

```
- machine Cray XC
    - cabinet row 1
        - cabinet 8
            - cage 0
                - blade 13
                    - asic 877
                        - nic 4660
                            - node nid03508
                                - MPI Rank 0
                                    0.00 Master thread
                                    1.57 OMP thread 1
                                    1.57 OMP thread 2
                                    1.57 OMP thread 3
                                    1.57 OMP thread 4
                                    1.57 OMP thread 5
                                - MPI Rank 1
                                    0.00 Master thread
                                    3.48 OMP thread 1
                                    3.48 OMP thread 2
                                    3.48 OMP thread 3
                                    3.48 OMP thread 4
                                    3.48 OMP thread 5
All (12 elements)
```

PERFORMANCE ANALYSIS

USING THE SCORE-P ECOSYSTEM

MOTIVATION

- Writing parallel code is hard
- Writing **fast/efficient** parallel code is even harder
- “Parallel” (multi core/node) performance factors
  - Partitioning / decomposition
    - ☞ Load balancing
  - Communication (i.e., message passing)
  - Multithreading
  - Core binding / NUMA
  - Synchronization / locking
  - I/O
    - ☞Parallel I/O matters

TUNING BASICS

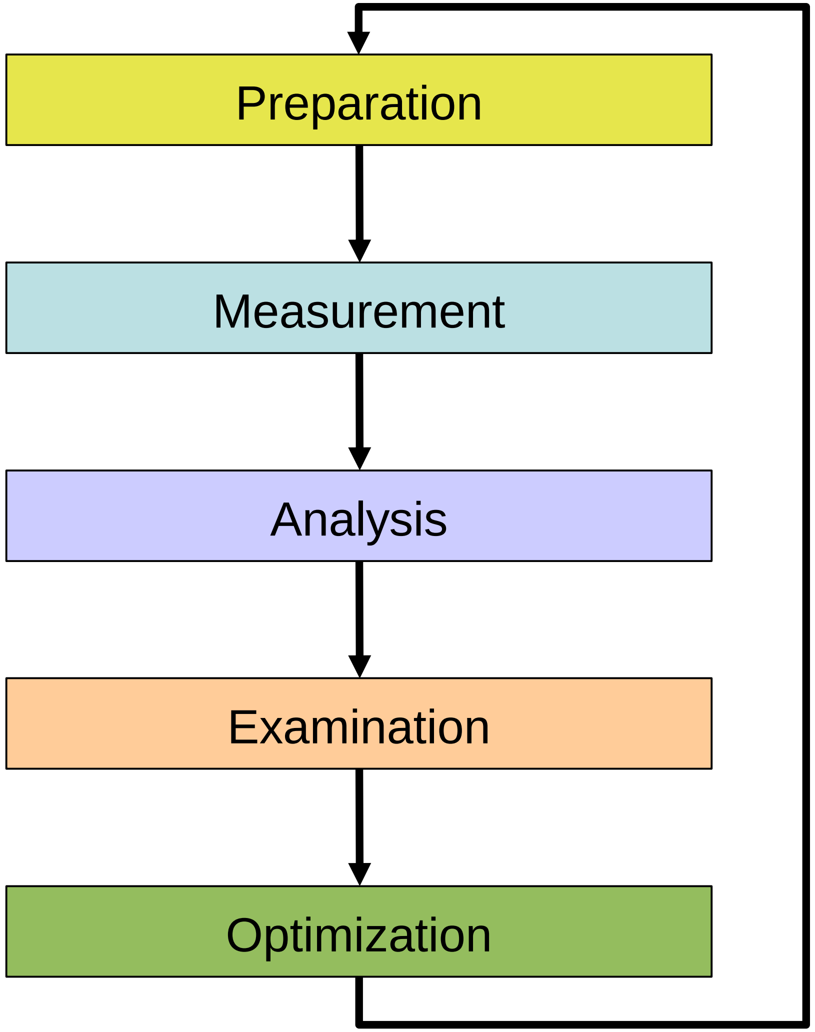
- Carefully set various tuning parameters
  - The right (parallel) algorithms and libraries
  - Compiler flags and directives
  - Correct machine usage (mapping and bindings)

    ☞Get the most performance before tuning!

- Measurement is better than guessing
  - To determine performance bottlenecks
  - To compare alternatives
  - To validate tuning decisions and optimizations

    ☞After each step!

PERFORMANCE ENGINEERING WORKFLOW



- Prepare application (with symbols), insert extra code (probes/hooks)
- Collection of data relevant to execution performance analysis
- Calculation of metrics, identification of performance metrics
- Presentation of results in an intuitive/understandable form
- Modifications intended to eliminate/reduce performance problems

THE 80/20 RULE

- Programs typically spend 80% of their time in 20% of the code
  - ☞ *Know what matters!*

- Developers typically spend 20% of their effort to get 80% of the total speedup possible for the application
  - ☞ *Know when to stop!*

- Don't optimize what does not matter
  - ☞ *Make the common case fast!*

PERFORMANCE MEASUREMENT

Two dimensions

When performance measurement is triggered

- **External trigger** (asynchronous)
  - **Sampling**
    - Trigger: Timer interrupt OR Hardware counters overflow
- **Internal trigger** (synchronous)
  - Code **instrumentation** (automatic or manual)

How performance data is recorded

- **Profile**
  - Summation of events over time
- **Trace**
  - Sequence of events over time

Score-P

Scalable performance measurement infrastructure for parallel codes

- Community-developed open-source
- Replaced tool-specific instrumentation and measurement components of partners
- http://www.score-p.org

TECHNISCHE UNIVERSITÄT DRESDEN

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

German Research School for Simulation Sciences

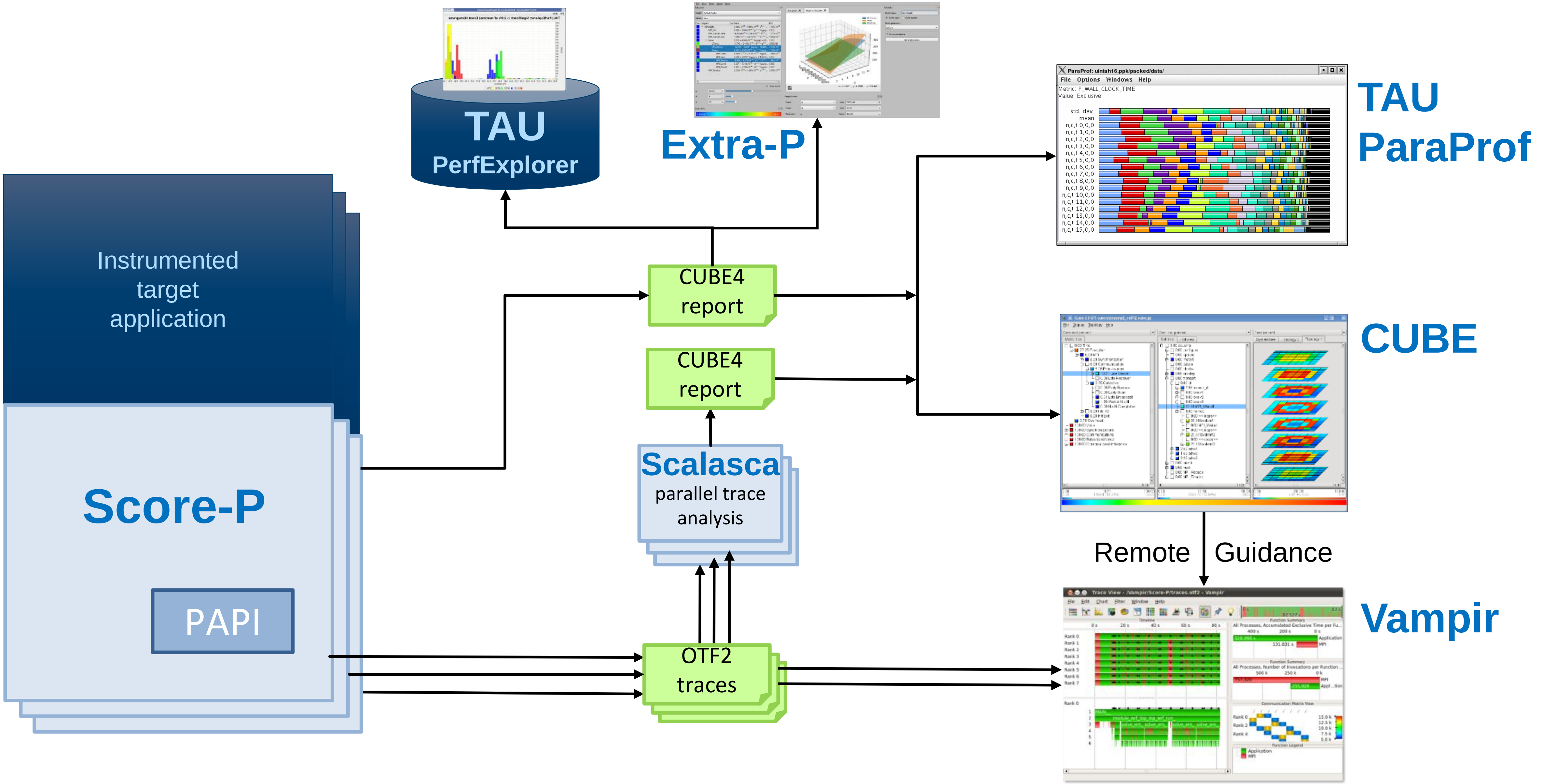
RWTH AACHEN UNIVERSITY

TUM Technische Universität München

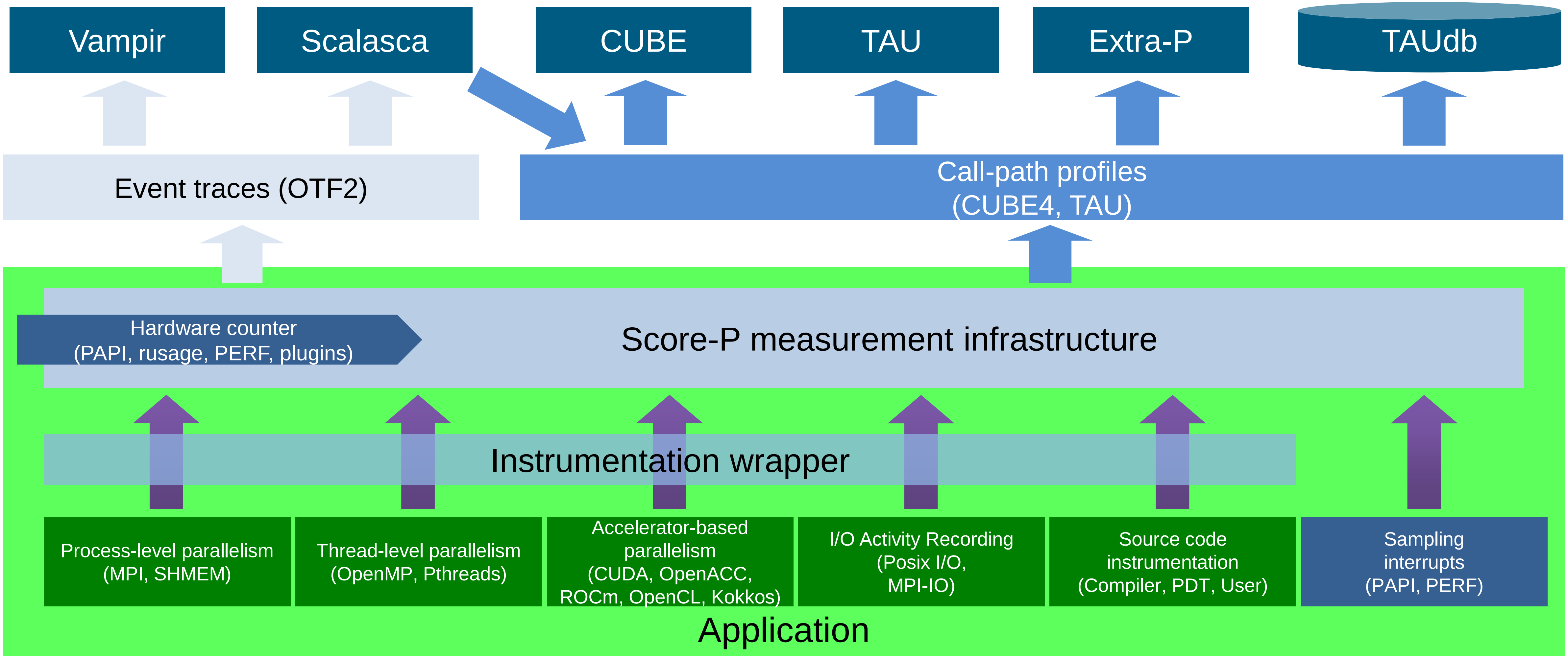
UNIVERSITY OF OREGON

Score-P TOOL ECOSYSTEM

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE



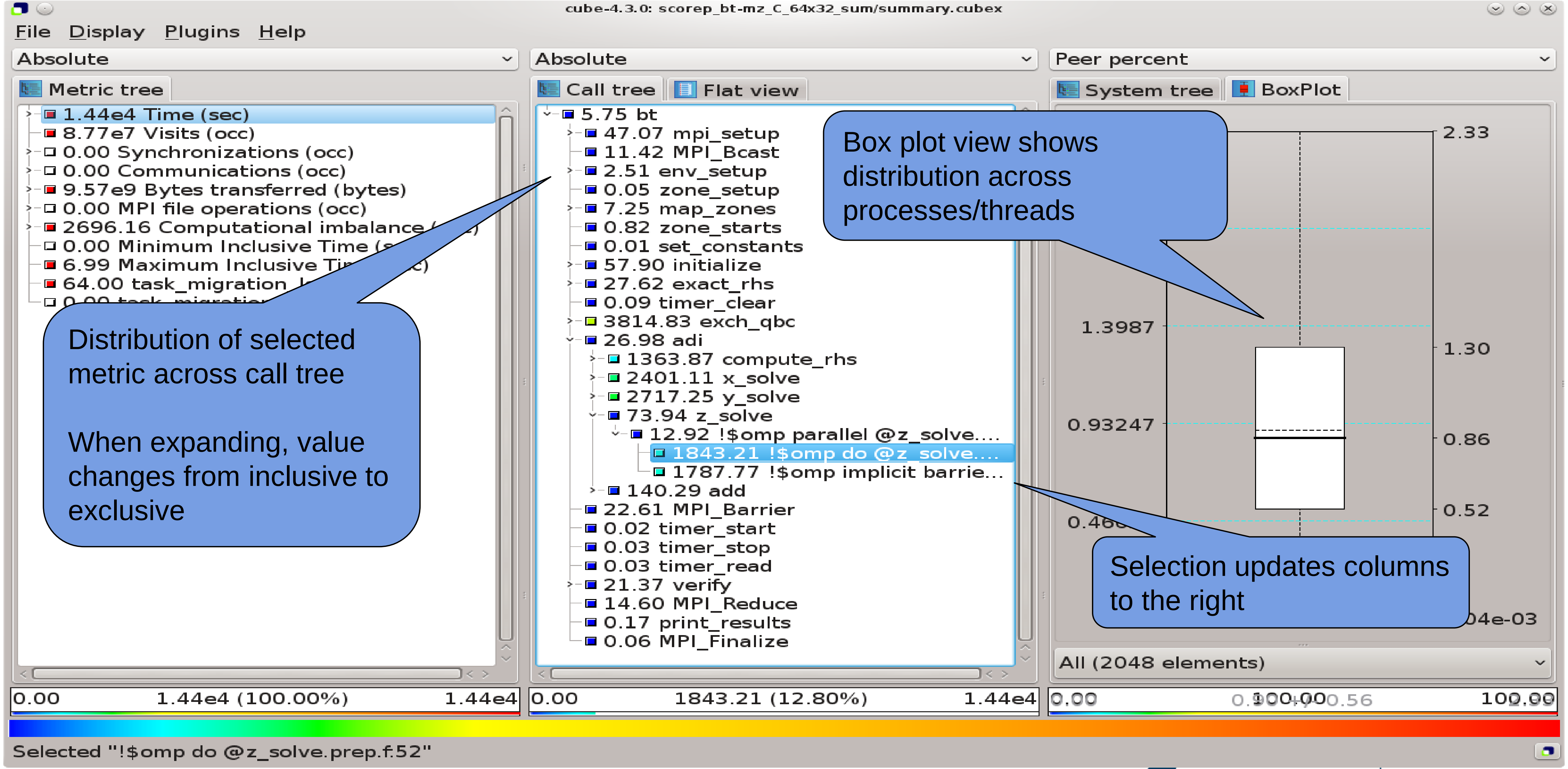
Score-P ARCHITECTURE



Score-P FUNCTIONALITY

- Provide typical functionality for HPC performance tools
- **Instrumentation** (various methods)
  - Multi-process paradigms (MPI, SHMEM)
  - Thread-parallel paradigms (OpenMP, POSIX threads)
  - Accelerator-based paradigms (OpenACC, CUDA, OpenCL. Kokkos)
  - **In any combination!**
- Flexible **measurement** without re-compilation:
  - Basic and advanced **profile** generation (⇨ CUBE4 format)
  - Event **trace** recording (⇨ OTF2 format)
- Highly scalable I/O functionality
- Support all fundamental concepts of partner’s tools

CUBE EXAMPLE



```
1.44e4 Time (sec)
8.77e7 Visits (occ)
0.00 Synchronizations (occ)
0.00 Communications (occ)
9.57e9 Bytes transferred (bytes)
0.00 MPI file operations (occ)
2696.16 Computational imbalance (   )
0.00 Minimum Inclusive Time (s
6.99 Maximum Inclusive Ti     c)
64.00 task_migration_l
0.00 task_migration
```

```
5.75 bt
  47.07 mpi_setup
  11.42 MPI_Bcast
  2.51 env_setup
  0.05 zone_setup
  7.25 map_zones
  0.82 zone_starts
  0.01 set_constants
  57.90 initialize
  27.62 exact_rhs
  0.09 timer_clear
  3814.83 exch_qbc
  26.98 adi
    1363.87 compute_rhs
    2401.11 x_solve
    2717.25 y_solve
    73.94 z_solve
      12.92 !$omp parallel @z_solve....
        1843.21 !$omp do @z_solve....
        1787.77 !$omp implicit barrie...
    140.29 add
  22.61 MPI_Barrier
  0.02 timer_start
  0.03 timer_stop
  0.03 timer_read
  21.37 verify
  14.60 MPI_Reduce
  0.17 print_results
  0.06 MPI_Finalize
```

0.00 1.44e4 (100.00%) 1.44e4

0.00 1843.21 (12.80%) 1.44e4

0.00 0.100.00+0.56 100.00

Selected "!$omp do @z_solve.prep.f.52"

Box plot view shows distribution across processes/threads

Distribution of selected metric across call tree

When expanding, value changes from inclusive to exclusive

Selection updates columns to the right

SCORE-P: ADVANCED FEATURES

- Measurement can be extensively configured via environment variables
- Allows for targeted measurements:
  - Selective recording
  - Phase profiling
  - Parameter-based profiling
  - …
- GPU support: CUDA, OpenACC, OpenCL, HIP, Kokkos, …
- Please ask us or see the user manual for details

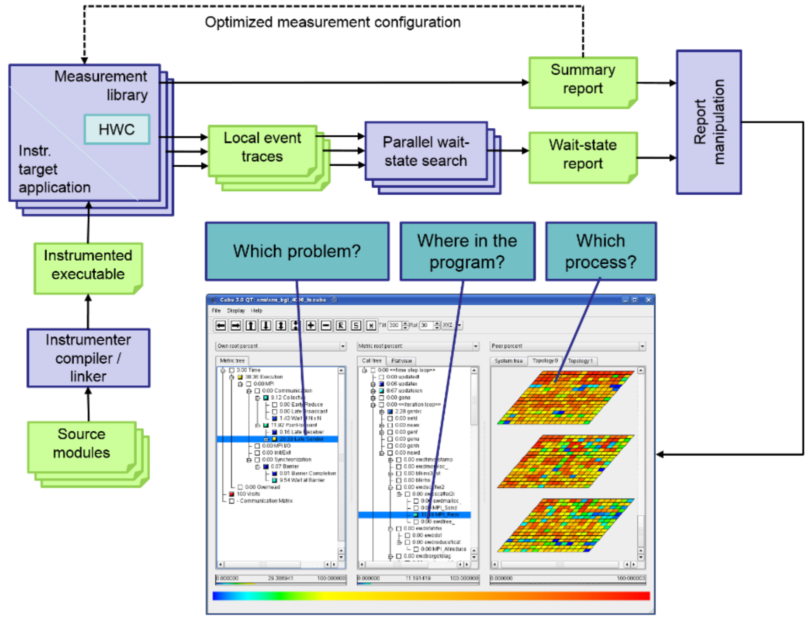
SCALASCA

scalasca

http://www.scalasca.org/

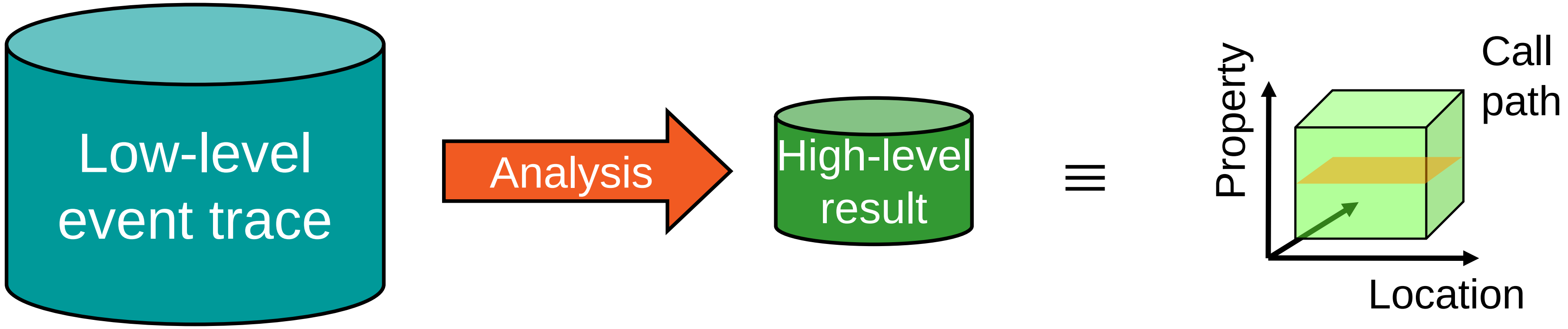
- Scalable Analysis of Large Scale Applications

- Approach
  - **Instrument** C, C++, and Fortran parallel applications (**with Score-P**)
  - Option 1: **scalable call-path profiling**
  - Option 2: **scalable event trace analysis**
    - **Collect** event traces
    - **Process trace in parallel**
      - Wait-state analysis
      - Delay and root-cause analysis
      - Critical path analysis
    - **Categorize and rank** results



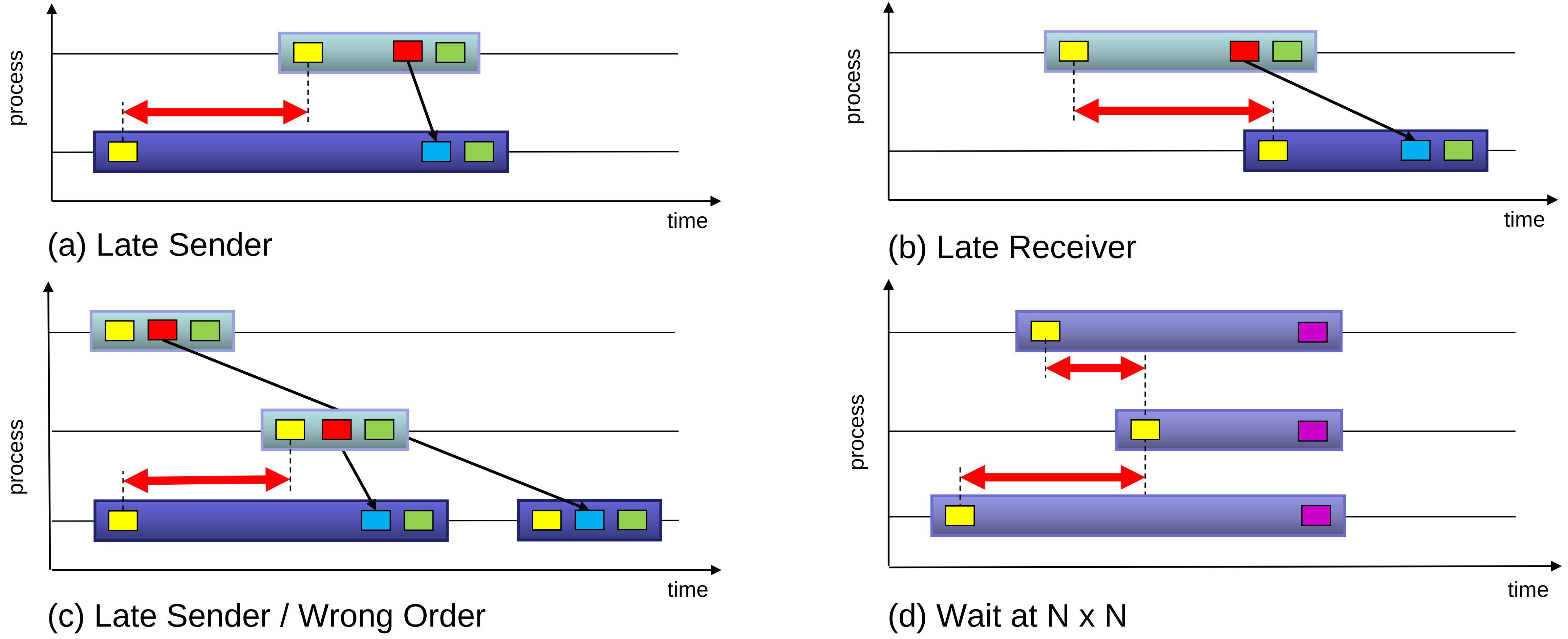
AUTOMATIC TRACE ANALYSIS

- Automatic search for patterns of inefficient behaviour
- Classification of behaviour & quantification of significance
- Identification of delays as root causes of inefficiencies



- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis exploits available memory & processors to deliver scalability

EXAMPLE MPI WAIT STATES



(a) Late Sender

(b) Late Receiver

(c) Late Sender / Wrong Order

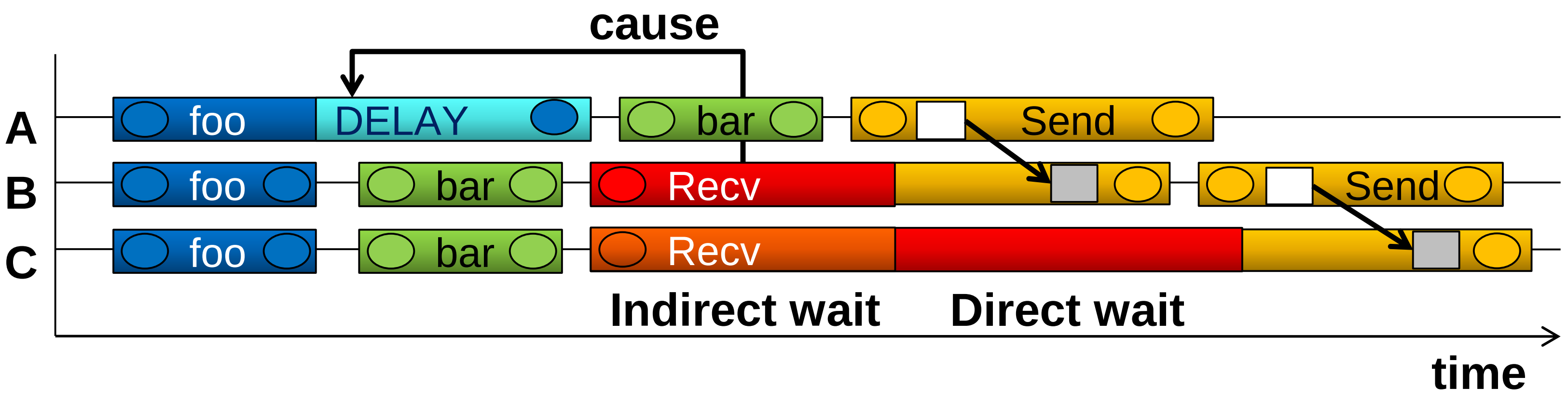
(d) Wait at N x N

🟨 ENTER 🟩 EXIT 🟥 SEND 🟦 RECV 🟪 COLLEXIT

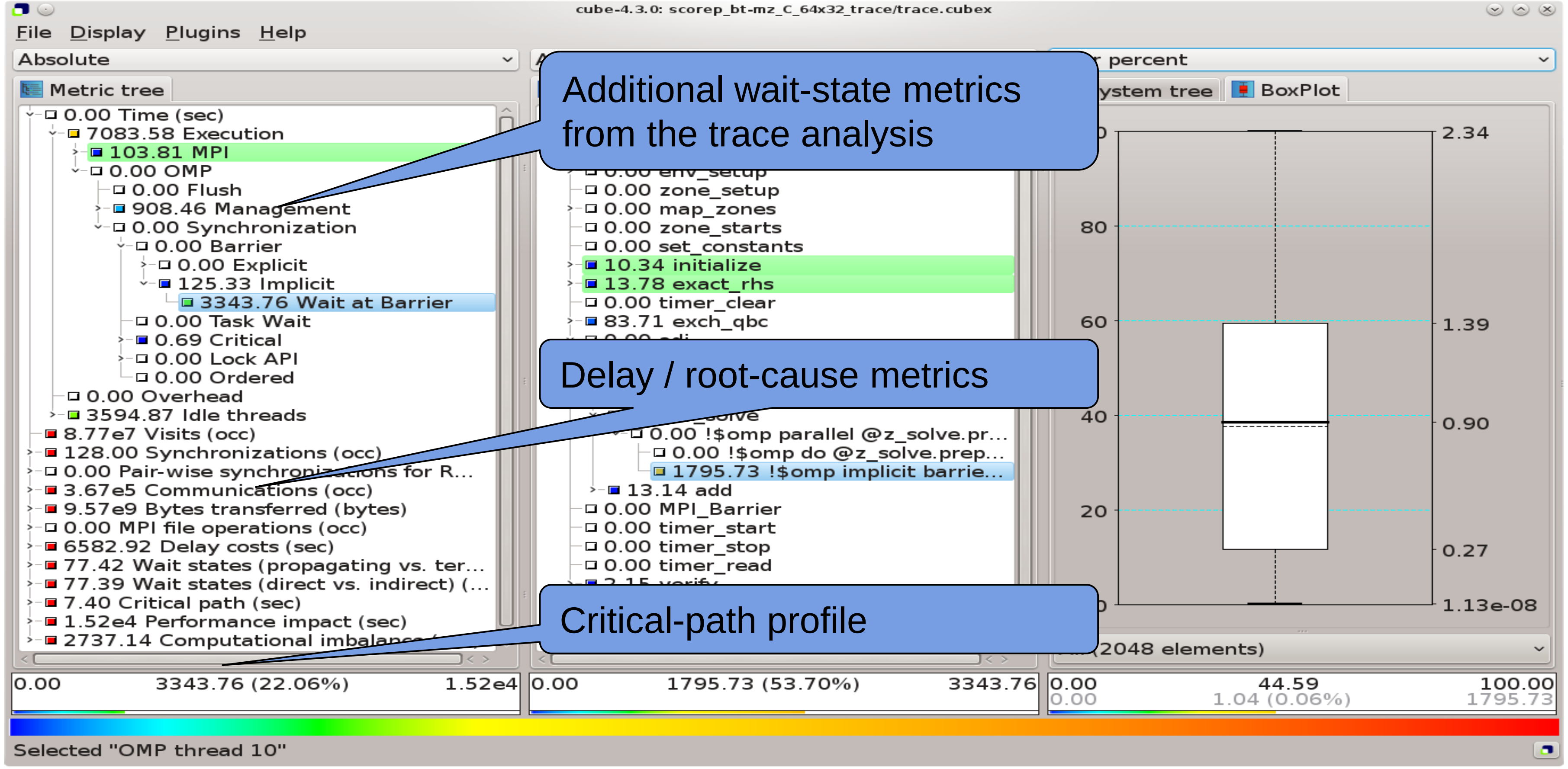
SCALASCA ROOT CAUSE ANALYSIS

- **Root-cause analysis**
  - Wait states typically caused by load or communication imbalances earlier in the program
  - Waiting time can also propagate (e.g., indirect waiting time)
  - Enhanced performance analysis to find the root cause of wait states

- **Approach**
  - Distinguish between direct and indirect waiting time
  - Identify call path/process combinations delaying other processes and causing first order waiting time
  - Identify original **delay**



SCALASCA TRACE ANALYSIS EXAMPLE



Metric tree:

```
0.00 Time (sec)
  7083.58 Execution
    103.81 MPI
    0.00 OMP
      0.00 Flush
      908.46 Management
      0.00 Synchronization
        0.00 Barrier
          0.00 Explicit
          125.33 Implicit
            3343.76 Wait at Barrier
        0.00 Task Wait
        0.69 Critical
        0.00 Lock API
        0.00 Ordered
  0.00 Overhead
  3594.87 Idle threads
8.77e7 Visits (occ)
128.00 Synchronizations (occ)
0.00 Pair-wise synchronizations for R...
3.67e5 Communications (occ)
9.57e9 Bytes transferred (bytes)
0.00 MPI file operations (occ)
6582.92 Delay costs (sec)
77.42 Wait states (propagating vs. ter...
77.39 Wait states (direct vs. indirect) (...
7.40 Critical path (sec)
1.52e4 Performance impact (sec)
2737.14 Computational imbalance (
```

Call tree:

```
0.00 zone_setup
0.00 map_zones
0.00 zone_starts
0.00 set_constants
10.34 initialize
13.78 exact_rhs
0.00 timer_clear
83.71 exch_qbc
    0.00 !$omp parallel @z_solve.pr...
      0.00 !$omp do @z_solve.prep...
      1795.73 !$omp implicit barrie...
  13.14 add
0.00 MPI_Barrier
0.00 timer_start
0.00 timer_stop
0.00 timer_read
```

```
0.00      3343.76 (22.06%)      1.52e4
0.00      1795.73 (53.70%)      3343.76
0.00      44.59                 100.00
0.00      1.04 (0.06%)          1795.73
```

Selected "OMP thread 10"

Additional wait-state metrics from the trace analysis

Delay / root-cause metrics

Critical-path profile

VAMPIR EVENT TRACE VISUALIZER

- Offline trace visualization for Score-Ps OTF2 trace files
- Visualization of MPI, OpenMP and application events:
  - All diagrams highly customizable (through context menus)
  - Large variety of displays for ANY part of the trace
- http://www.vampir.eu

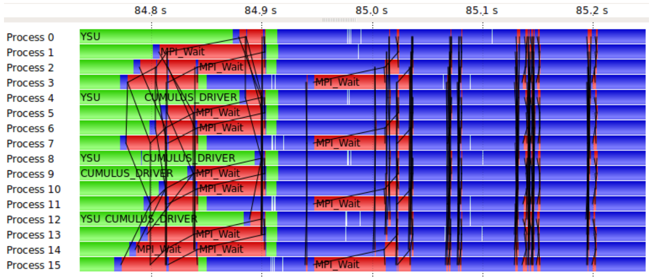
- Advantage:
  - Detailed view of dynamic application behavior
- Disadvantage:
  - Completely manual analysis
  - Too many details can hide the relevant parts

EVENT TRACE VISUALIZATION WITH VAMPIR

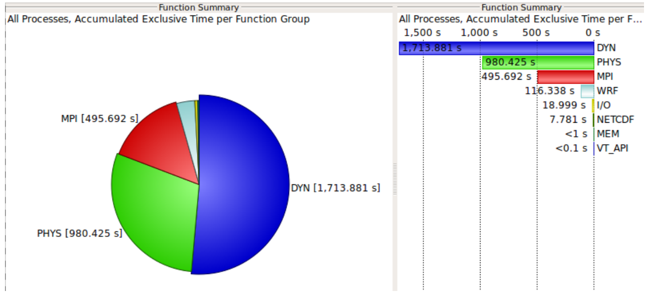
- Visualization of dynamic runtime behaviour at any level of detail along with statistics and performance metrics
- Alternative and supplement to automatic analysis

- **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 communication along a time axis



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



VAMPIR PERFORMANCE CHARTS

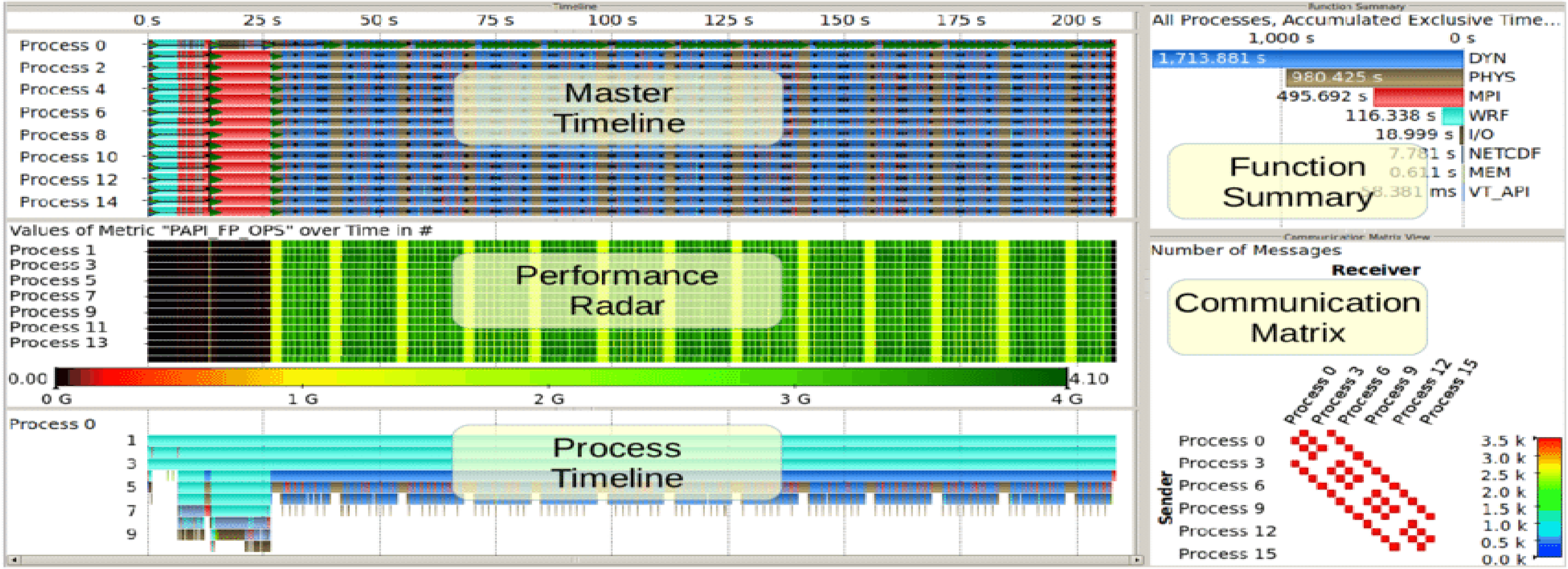
Timeline Charts

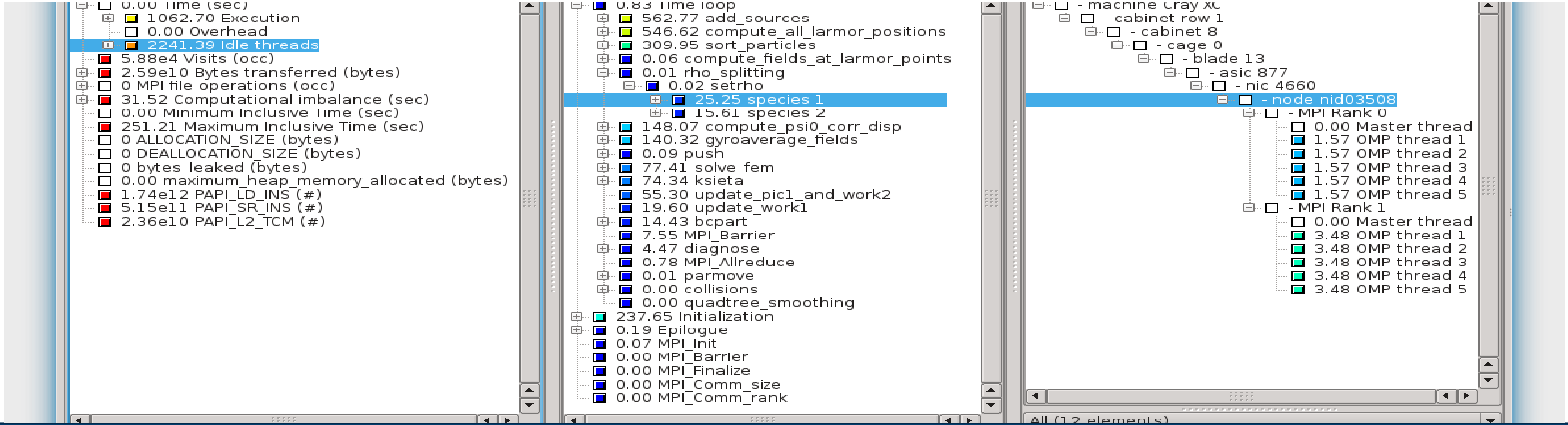
Master Timeline	➡	*all threads' activities*
Process Timeline	➡	*single thread's activities*
Summary Timeline	➡	*all threads' function call statistics*
Performance Radar	➡	*all threads' performance metrics*
Counter Data Timeline	➡	*single threads' performance metrics*
I/O Timeline	➡	*all threads' I/O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

Summary Charts

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

VAMPIR DISPLAYS





```
0.00 Time (sec)
  1062.70 Execution
  0.00 Overhead
  2241.39 Idle threads
5.88e4 Visits (occ)
2.59e10 Bytes transferred (bytes)
0 MPI file operations (occ)
31.52 Computational imbalance (sec)
0.00 Minimum Inclusive Time (sec)
251.21 Maximum Inclusive Time (sec)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
1.74e12 PAPI_LD_INS (#)
5.15e11 PAPI_SR_INS (#)
2.36e10 PAPI_L2_TCM (#)
```

```
0.83 Time loop
  562.77 add_sources
  546.62 compute_all_larmor_positions
  309.95 sort_particles
  0.06 compute_fields_at_larmor_points
  0.01 rho_splitting
    0.02 setrho
      25.25 species 1
      15.61 species 2
  148.07 compute_psi0_corr_disp
  140.32 gyroaverage_fields
  0.09 push
  77.41 solve_fem
  74.34 ksieta
  55.30 update_pic1_and_work2
  19.60 update_work1
  14.43 bcpart
  7.55 MPI_Barrier
  4.47 diagnose
  0.78 MPI_Allreduce
  0.01 parmove
  0.00 collisions
  0.00 quadtree_smoothing
237.65 Initialization
0.19 Epilogue
0.07 MPI_Init
0.00 MPI_Barrier
0.00 MPI_Finalize
0.00 MPI_Comm_size
0.00 MPI_Comm_rank
```

```
- machine Cray XC
  - cabinet row 1
    - cabinet 8
      - cage 0
        - blade 13
          - asic 877
            - nic 4660
              - node nid03508
                - MPI Rank 0
                  0.00 Master thread
                  1.57 OMP thread 1
                  1.57 OMP thread 2
                  1.57 OMP thread 3
                  1.57 OMP thread 4
                  1.57 OMP thread 5
                - MPI Rank 1
                  0.00 Master thread
                  3.48 OMP thread 1
                  3.48 OMP thread 2
                  3.48 OMP thread 3
                  3.48 OMP thread 4
                  3.48 OMP thread 5
All (12 elements)
```

TOOLS DEMO: BT-MZ WITH SCORE-P

TYPICAL PERFORMANCE ANALYSIS PROCEDURE

- Do I have a performance problem at all?
  - Time / speedup / scalability measurements
- ***What*** is the key bottleneck (computation / communication)?
  - MPI / OpenMP / flat profiling
- ***Where*** is the key bottleneck?
  - Call-path profiling, detailed basic block profiling
- ***Why*** is it there?
  - Hardware counter analysis
  - Trace selected parts (to keep trace size manageable)
- Does the code have scalability problems?
  - Load imbalance analysis, compare profiles at various sizes function-by-function, performance modeling

WHAT IS THE KEY BOTTLENECK?

- Generate flat MPI profile using Score-P/Scalasca
  - Only requires re-linking
  - Low runtime overhead

- Provides detailed information on MPI usage
  - How much time is spent in which operation?
  - How often is each operation called?
  - How much data was transferred?

- Limitations:
  - Computation on non-master threads and outside of MPI_Init/MPI_Finalize scope ignored

FLAT MPI PROFILE: RECIPE

1. Prefix your ***link command*** with

   ```
   “scorep --nocompiler”
   ```

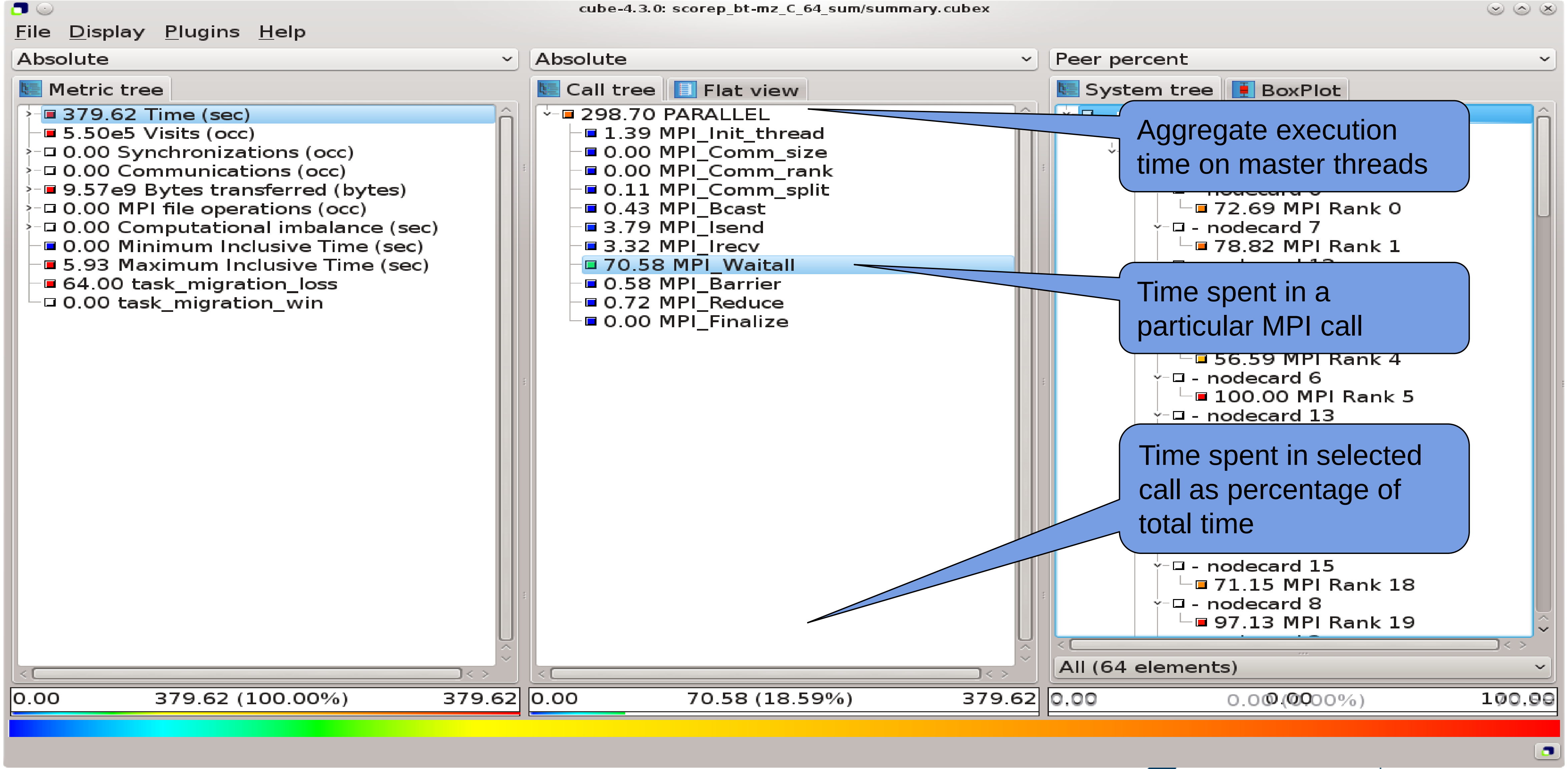
2. Prefix your MPI ***launch command*** with

   ```
   “scalasca -analyze”
   ```

3. After execution, examine analysis results using

   ```
   “scalasca -examine scorep_<title>”
   ```

FLAT MPI PROFILE: EXAMPLE (CONT.)



```
379.62 Time (sec)
5.50e5 Visits (occ)
0.00 Synchronizations (occ)
0.00 Communications (occ)
9.57e9 Bytes transferred (bytes)
0.00 MPI file operations (occ)
0.00 Computational imbalance (sec)
0.00 Minimum Inclusive Time (sec)
5.93 Maximum Inclusive Time (sec)
64.00 task_migration_loss
0.00 task_migration_win
```

```
298.70 PARALLEL
  1.39 MPI_Init_thread
  0.00 MPI_Comm_size
  0.00 MPI_Comm_rank
  0.11 MPI_Comm_split
  0.43 MPI_Bcast
  3.79 MPI_Isend
  3.32 MPI_Irecv
  70.58 MPI_Waitall
  0.58 MPI_Barrier
  0.72 MPI_Reduce
  0.00 MPI_Finalize
```

```
    72.69 MPI Rank 0
  - nodecard 7
    78.82 MPI Rank 1
    56.59 MPI Rank 4
  - nodecard 6
    100.00 MPI Rank 5
  - nodecard 13
  - nodecard 15
    71.15 MPI Rank 18
  - nodecard 8
    97.13 MPI Rank 19
```

```
0.00   379.62 (100.00%)   379.62
0.00   70.58 (18.59%)   379.62
0.00   0.00 (0.00%)   100.00
```

Aggregate execution time on master threads

Time spent in a particular MPI call

Time spent in selected call as percentage of total time

WHERE IS THE KEY BOTTLENECK?

- Generate call-path profile using Score-P/Scalasca
  - Requires re-compilation
  - Runtime overhead depends on application characteristics
  - Typically needs some care setting up a good measurement configuration
    - Filtering
    - Selective instrumentation

- Option 1 (recommended for beginners):
  Automatic compiler-based instrumentation

- Option 2 (for in-depth analysis):
  Manual instrumentation of interesting phases, routines, loops

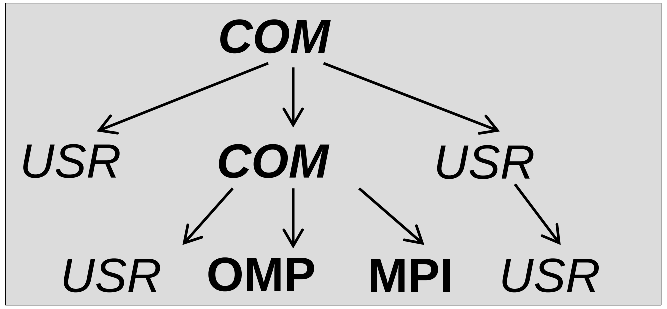
CALL-PATH PROFILE: RECIPE

1. Prefix your ***compile & link commands*** with
   “`scorep`”
2. Prefix your MPI ***launch command*** with
   “`scalasca -analyze`”
3. After execution, compare overall runtime with uninstrumented run to determine overhead
4. If overhead is too high
   1. Score measurement using
      “`scalasca -examine -s scorep_<title>`”
   2. Prepare filter file
   3. Re-run measurement with filter applied using prefix
      “`scalasca –analyze –f <filter_file>`”
5. After execution, examine analysis results using
   “scalasca -examine scorep_<*title*>”

CALL-PATH PROFILE: EXAMPLE (CONT.)

```
% scalasca -examine -s scorep_myprog_Ppnxt_sum
scorep-score -r ./scorep_myprog_Ppnxt_sum/profile.cubex
INFO: Score report written to ./scorep_myprog_Ppnxt_sum/scorep.score
```

- Estimates trace buffer requirements
- Allows to identify canditate functions for filtering
  - ☞Computational routines with high visit count and low time-per-visit ratio
- Region/call-path classification
  - MPI (pure MPI library functions)
  - OMP (pure OpenMP functions/regions)
  - USR (user-level source local computation
  - COM (“combined” USR + OpeMP/MPI)
  - ANY/ALL (aggregate of all region types)



CALL-PATH PROFILE: EXAMPLE (CONT.)

```
% less scorep_myprog_Ppnxt_sum/scorep.score
Estimated aggregate size of event trace:                   162GB
Estimated requirements for largest trace buffer (max_buf): 2758MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       2822MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=2822MB to avoid
 intermediate flushes or reduce requirements using USR regions
 filters.)

flt type     max_buf[B]          visits  time[s] time[%] time/      region
                                                         visit[us]
     ALL 2,891,417,902 6,662,521,083 36581.51   100.0       5.49  ALL
     USR 2,858,189,854 6,574,882,113 13618.14    37.2       2.07  USR
     OMP    54,327,600    86,353,920 22719.78    62.1     263.10  OMP
     MPI       676,342       550,010   208.98     0.6     379.96  MPI
     COM       371,930       735,040    34.61     0.1      47.09  COM

     USR   921,918,660 2,110,313,472  3290.11     9.0       1.56  matmul_sub
     USR   921,918,660 2,110,313,472  5914.98    16.2       2.80  binvcrhs
     USR   921,918,660 2,110,313,472  3822.64    10.4       1.81  matvec_sub
     USR    41,071,134    87,475,200   358.56     1.0       4.10  lhsinit
     USR    41,071,134    87,475,200   145.42     0.4       1.66  binvrhs
     USR    29,194,256    68,892,672    86.15     0.2       1.25  exact_solution
     OMP     3,280,320     3,293,184    15.81     0.0       4.80  !$omp parallel
     [...]
```

CALL-PATH PROFILE: FILTERING

- In this example, the 6 most fequently called routines are of type USR
- These routines contribute around 35% of total time
  - However, much of that is most likely measurement overhead
    - Frequently executed
    - Time-per-visit ratio in the order of a few microseconds

☞Avoid measurements to reduce the overhead

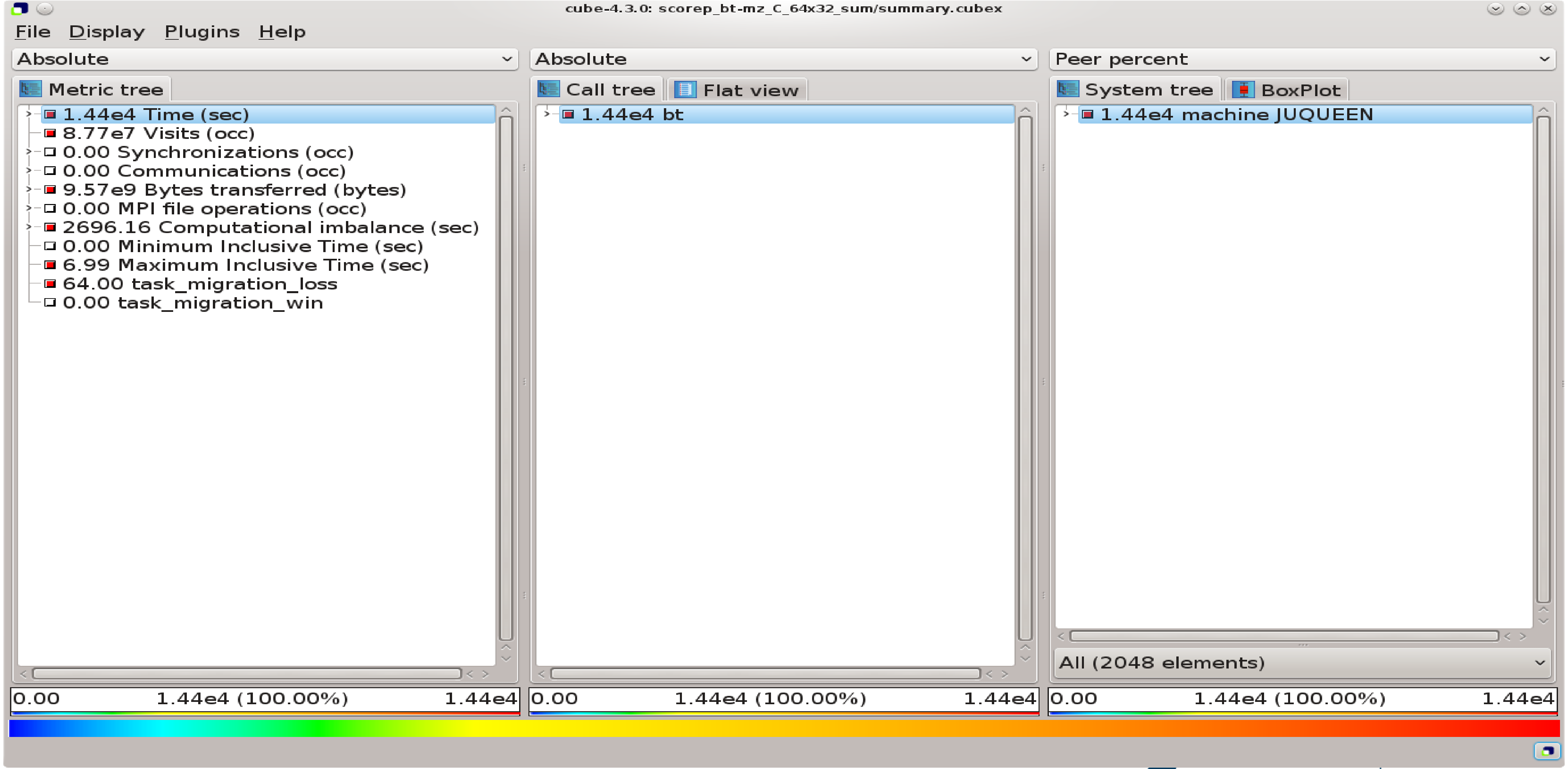
☞List routines to be filtered in simple text file

FILTERING: EXAMPLE

```
% cat filter.txt
SCOREP_REGION_NAMES_BEGIN
    EXCLUDE
        binvcrhs
        matmul_sub
        matvec_sub
        binvrhs
        lhsinit
        exact_solution
SCOREP_REGION_NAMES_END
```

- Score-P filtering files support
  - Wildcards (shell globs)
  - Blacklisting
  - Whitelisting
  - Filtering based on filenames

CALL-PATH PROFILE: EXAMPLE (CONT.)



- 1.44e4 Time (sec)
- 8.77e7 Visits (occ)
- 0.00 Synchronizations (occ)
- 0.00 Communications (occ)
- 9.57e9 Bytes transferred (bytes)
- 0.00 MPI file operations (occ)
- 2696.16 Computational imbalance (sec)
- 0.00 Minimum Inclusive Time (sec)
- 6.99 Maximum Inclusive Time (sec)
- 64.00 task_migration_loss
- 0.00 task_migration_win

1.44e4 bt

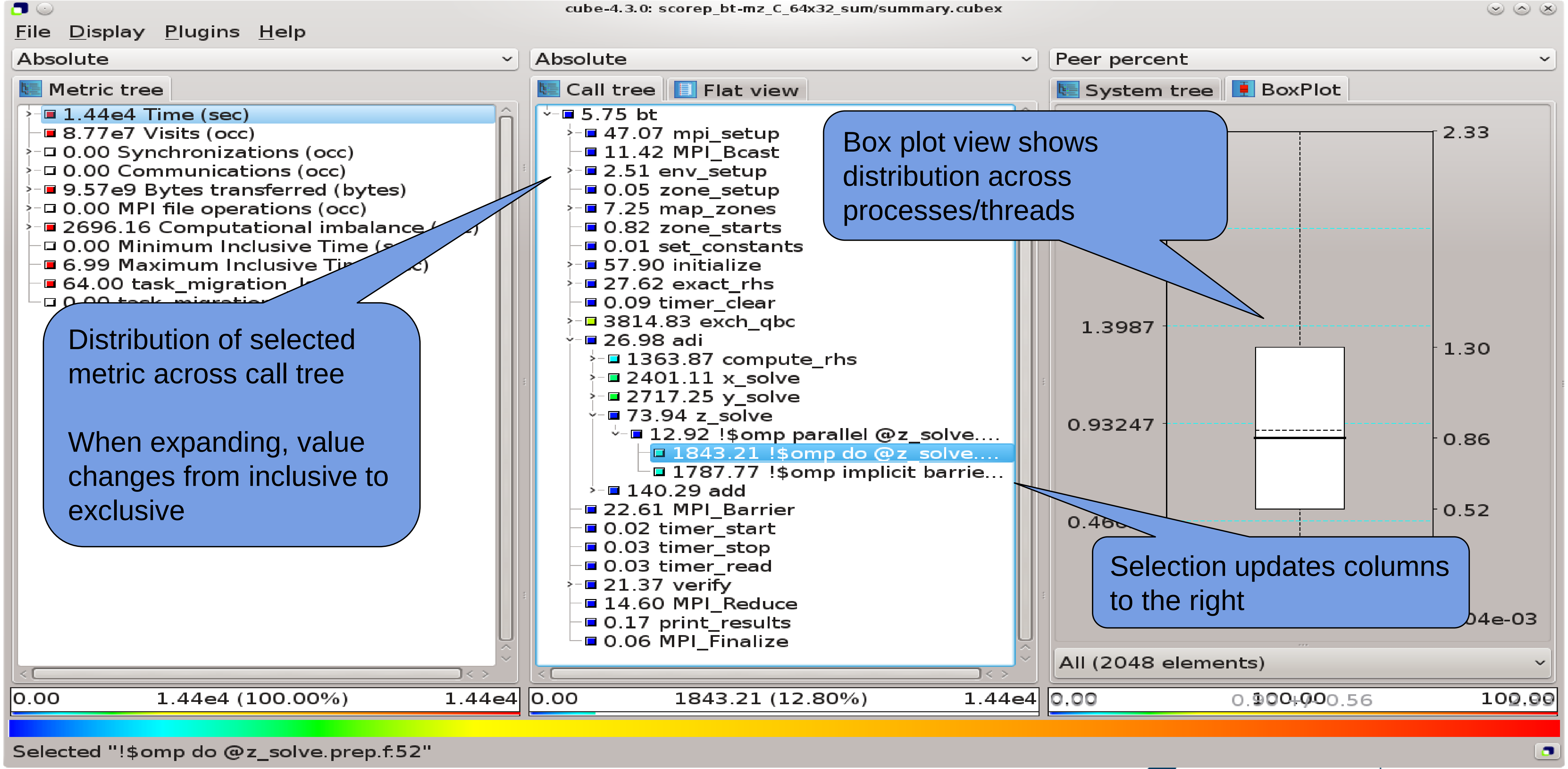
1.44e4 machine JUQUEEN

0.00 1.44e4 (100.00%) 1.44e4

0.00 1.44e4 (100.00%) 1.44e4

0.00 1.44e4 (100.00%) 1.44e4

CALL-PATH PROFILE: EXAMPLE (CONT.)



cube-4.3.0: scorep_bt-mz_C_64x32_sum/summary.cubex

```
1.44e4 Time (sec)
8.77e7 Visits (occ)
0.00 Synchronizations (occ)
0.00 Communications (occ)
9.57e9 Bytes transferred (bytes)
0.00 MPI file operations (occ)
2696.16 Computational imbalance (    )
0.00 Minimum Inclusive Time (s
6.99 Maximum Inclusive Ti       c)
64.00 task_migration_l
0.00 task_migratio
```

```
5.75 bt
  47.07 mpi_setup
  11.42 MPI_Bcast
  2.51 env_setup
  0.05 zone_setup
  7.25 map_zones
  0.82 zone_starts
  0.01 set_constants
  57.90 initialize
  27.62 exact_rhs
  0.09 timer_clear
  3814.83 exch_qbc
  26.98 adi
    1363.87 compute_rhs
    2401.11 x_solve
    2717.25 y_solve
    73.94 z_solve
      12.92 !$omp parallel @z_solve....
        1843.21 !$omp do @z_solve....
        1787.77 !$omp implicit barrie...
    140.29 add
  22.61 MPI_Barrier
  0.02 timer_start
  0.03 timer_stop
  0.03 timer_read
  21.37 verify
  14.60 MPI_Reduce
  0.17 print_results
  0.06 MPI_Finalize
```

```
0.00    1.44e4 (100.00%)    1.44e4
0.00    1843.21 (12.80%)    1.44e4
0.00    100.00  0.90 +/- 0.56    100.00
```

Selected "!$omp do @z_solve.prep.f:52"

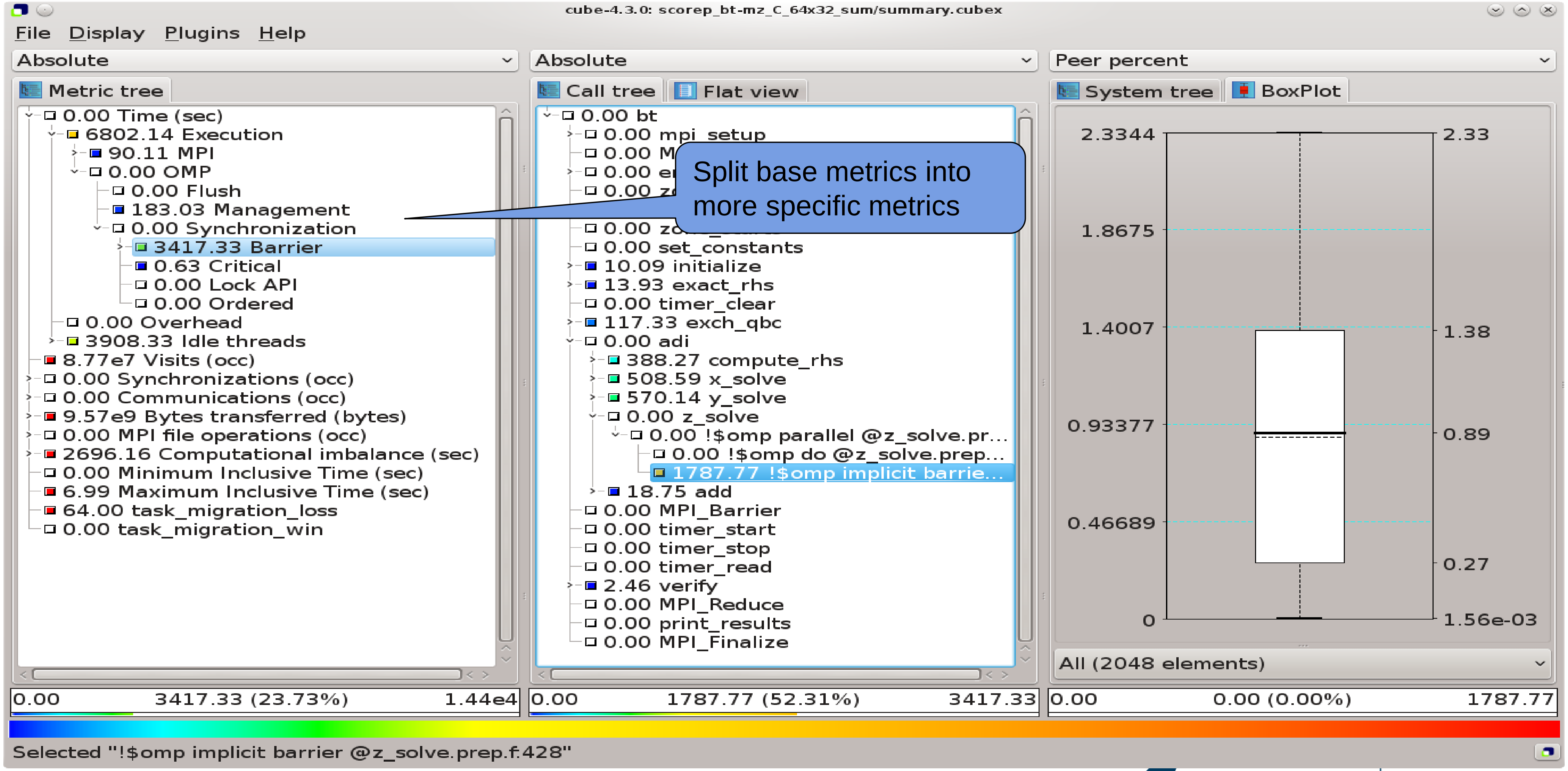
Box plot view shows distribution across processes/threads

Distribution of selected metric across call tree

When expanding, value changes from inclusive to exclusive

Selection updates columns to the right

CALL-PATH PROFILE: EXAMPLE (CONT.)



cube-4.3.0: scorep_bt-mz_C_64x32_sum/summary.cubex

```
□ 0.00 Time (sec)
  □ 6802.14 Execution
    □ 90.11 MPI
    □ 0.00 OMP
      □ 0.00 Flush
      □ 183.03 Management
      □ 0.00 Synchronization
        □ 3417.33 Barrier
        □ 0.63 Critical
        □ 0.00 Lock API
        □ 0.00 Ordered
  □ 0.00 Overhead
  □ 3908.33 Idle threads
□ 8.77e7 Visits (occ)
□ 0.00 Synchronizations (occ)
□ 0.00 Communications (occ)
□ 9.57e9 Bytes transferred (bytes)
□ 0.00 MPI file operations (occ)
□ 2696.16 Computational imbalance (sec)
□ 0.00 Minimum Inclusive Time (sec)
□ 6.99 Maximum Inclusive Time (sec)
□ 64.00 task_migration_loss
□ 0.00 task_migration_win
```

```
□ 0.00 bt
  □ 0.00 mpi_setup
  □ 0.00 M
  □ 0.00 e
  □ 0.00 z
  □ 0.00 zo
  □ 0.00 set_constants
  □ 10.09 initialize
  □ 13.93 exact_rhs
  □ 0.00 timer_clear
  □ 117.33 exch_qbc
  □ 0.00 adi
    □ 388.27 compute_rhs
    □ 508.59 x_solve
    □ 570.14 y_solve
    □ 0.00 z_solve
      □ 0.00 !$omp parallel @z_solve.pr...
        □ 0.00 !$omp do @z_solve.prep...
        □ 1787.77 !$omp implicit barrie...
    □ 18.75 add
  □ 0.00 MPI_Barrier
  □ 0.00 timer_start
  □ 0.00 timer_stop
  □ 0.00 timer_read
  □ 2.46 verify
  □ 0.00 MPI_Reduce
  □ 0.00 print_results
  □ 0.00 MPI_Finalize
```

```
0.00    3417.33 (23.73%)    1.44e4
0.00    1787.77 (52.31%)    3417.33
0.00    0.00 (0.00%)    1787.77
```

Selected "!$omp implicit barrier @z_solve.prep.f:428"

Split base metrics into more specific metrics

WHY IS THE BOTTLENECK THERE?

- This is highly application dependent!
- Might require additional measurements
  - Hardware-counter analysis
    - CPU utilization
    - Cache behavior
  - Selective instrumentation
  - Automatic/manual event trace analysis

HARDWARE COUNTERS

- Counters: set of registers that count processor events, e.g. floating point operations or cycles
- Number of registers, counters and simultaneously measurable events vary between platforms
- Can be measured by:
  - perf:
    - Integrated in Linux since Kernel 2.6.31
    - Library and CLI
  - LIKWID:
    - Direct access to MSRs (requires Kernel module)
    - Consists of multiple tools and an API
  - PAPI (Performance API)

PAPI

- Portable API: Uses the same routines to access counters across all supported architectures
- Used by most performance analysis tools

- High-level interface:
  - Predefined standard events, e.g. PAPI_FP_OPS
  - Availability and definition of events varies between platforms
  - List of available counters: `papi_avail (-d)`
- Low-level interface:
  - Provides access to all machine specific counters
  - Non-portable
  - More flexible
  - List of available counters: `papi_native_avail`

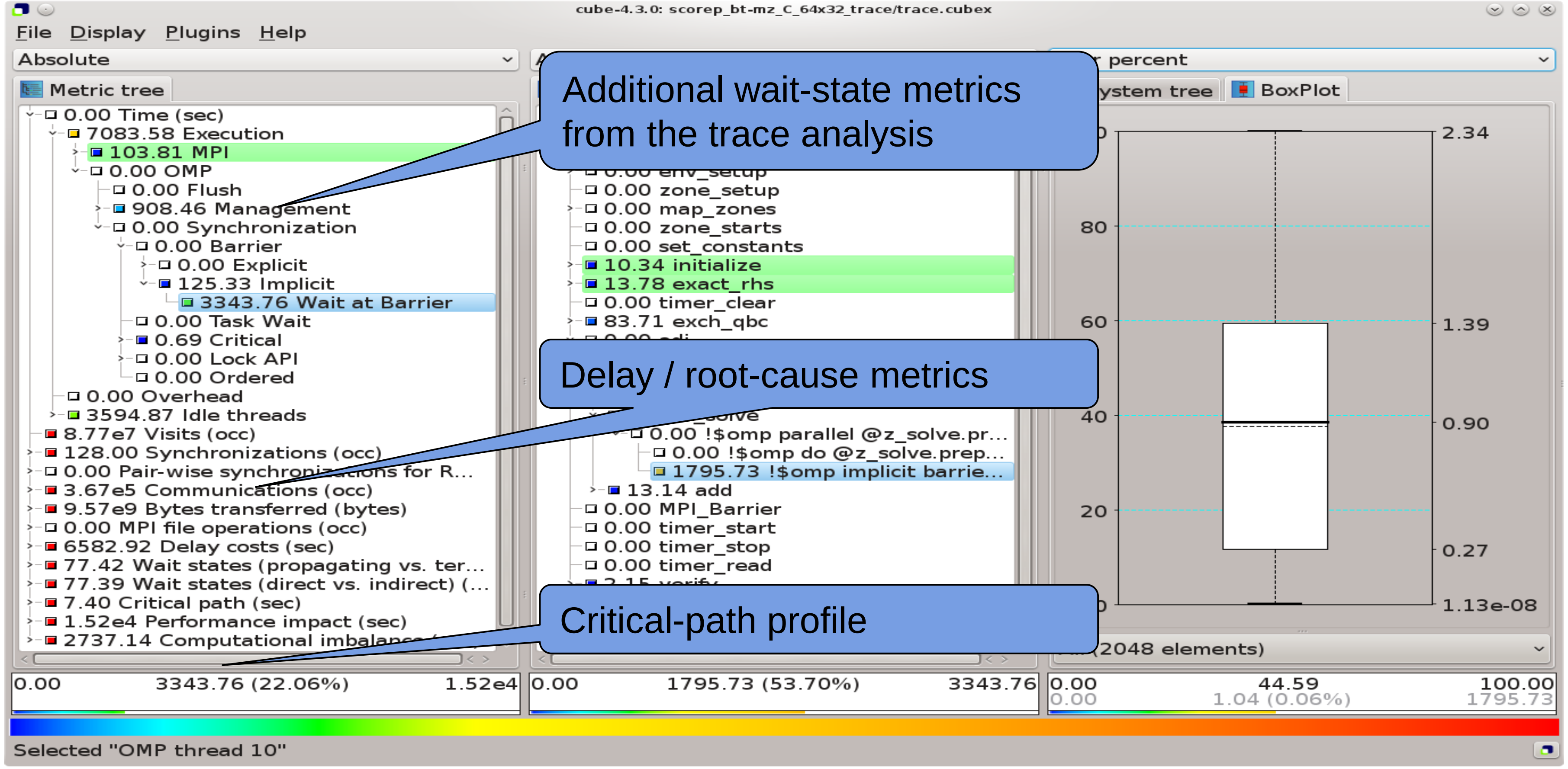
TRACE GENERATION & ANALYSIS W/ SCALASCA

- Enable trace collection & analysis using “`-t`” option of “`scalasca -analyze`”:

```
##########################
##  In the job script:  ##
##########################

module load ENV Score-P Scalasca
export SCOREP_TOTAL_MEMORY=120MB   # Consult score report
scalasca -analyze -f filter.txt -t \
    srun -n n [...] ./myprog
```

- **ATTENTION:**
  - Traces can quickly become extremely large!
  - Remember to use proper filtering, selective instrumentation, and Score-P memory specification
  - Before flooding the file system, **ask us for assistance!**

SCALASCA TRACE ANALYSIS EXAMPLE



cube-4.3.0: scorep_bt-mz_C_64x32_trace/trace.cubex

```
0.00 Time (sec)
  7083.58 Execution
    103.81 MPI
    0.00 OMP
      0.00 Flush
      908.46 Management
      0.00 Synchronization
        0.00 Barrier
          0.00 Explicit
          125.33 Implicit
            3343.76 Wait at Barrier
        0.00 Task Wait
        0.69 Critical
        0.00 Lock API
        0.00 Ordered
  0.00 Overhead
  3594.87 Idle threads
8.77e7 Visits (occ)
128.00 Synchronizations (occ)
0.00 Pair-wise synchronizations for R...
3.67e5 Communications (occ)
9.57e9 Bytes transferred (bytes)
0.00 MPI file operations (occ)
6582.92 Delay costs (sec)
77.42 Wait states (propagating vs. ter...
77.39 Wait states (direct vs. indirect) (...
7.40 Critical path (sec)
1.52e4 Performance impact (sec)
2737.14 Computational imbalance
```

```
0.00 env_setup
0.00 zone_setup
0.00 map_zones
0.00 zone_starts
0.00 set_constants
10.34 initialize
13.78 exact_rhs
0.00 timer_clear
83.71 exch_qbc
0.00 adi
    _solve
  0.00 !$omp parallel @z_solve.pr...
    0.00 !$omp do @z_solve.prep...
    1795.73 !$omp implicit barrie...
  13.14 add
0.00 MPI_Barrier
0.00 timer_start
0.00 timer_stop
0.00 timer_read
2.15 verify
```

(2048 elements)

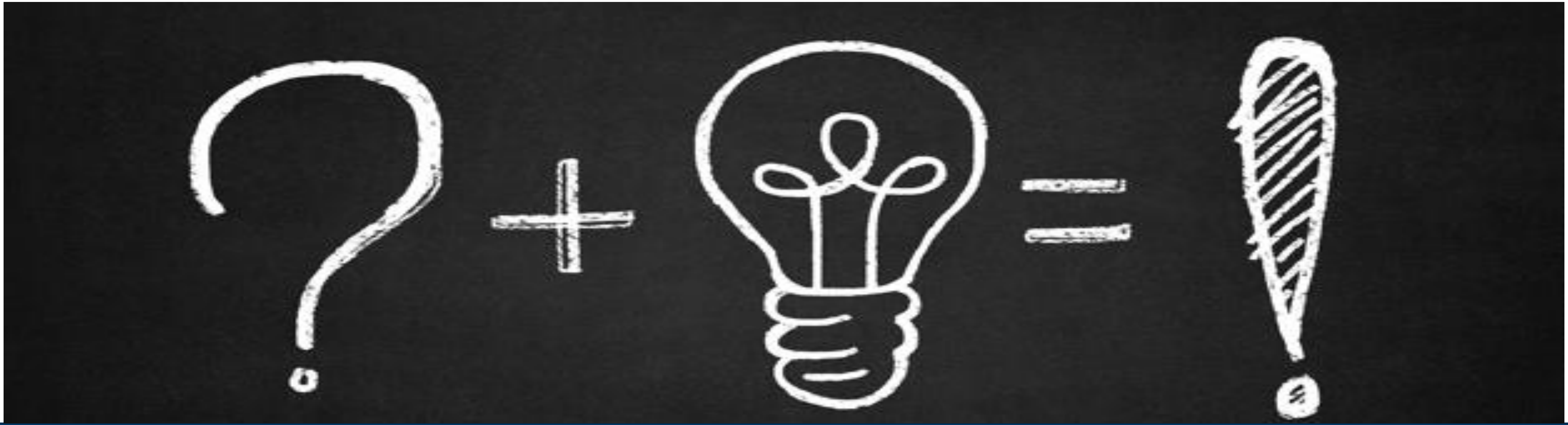
```
0.00     3343.76 (22.06%)     1.52e4
0.00     1795.73 (53.70%)     3343.76
0.00     44.59                100.00
0.00     1.04 (0.06%)         1795.73
```

Selected "OMP thread 10"

Additional wait-state metrics from the trace analysis

Delay / root-cause metrics

Critical-path profile



QUESTIONS