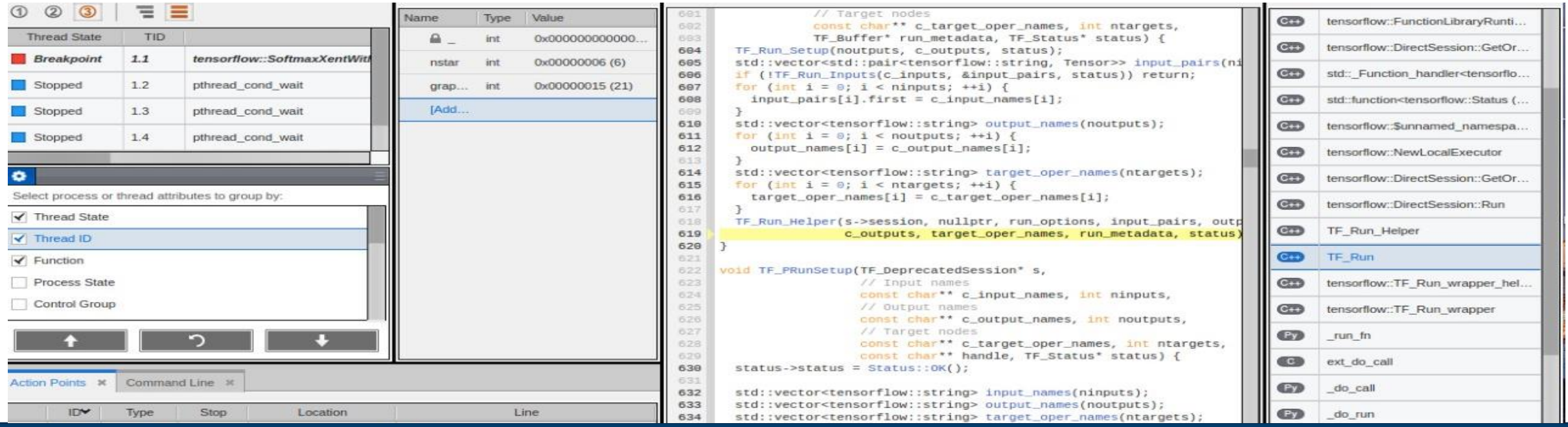
MPI/OPENMP COURSE – MUST

AUGUST 15, 2024 I MICHAEL KNOBLOCH I M.KNOBLOCH@FZ-JUELICH.DE



MUST – MPI CORRECTNESS CHECKER

HOW MANY ISSUES CAN YOU SPOT?

```c
#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?

- 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)

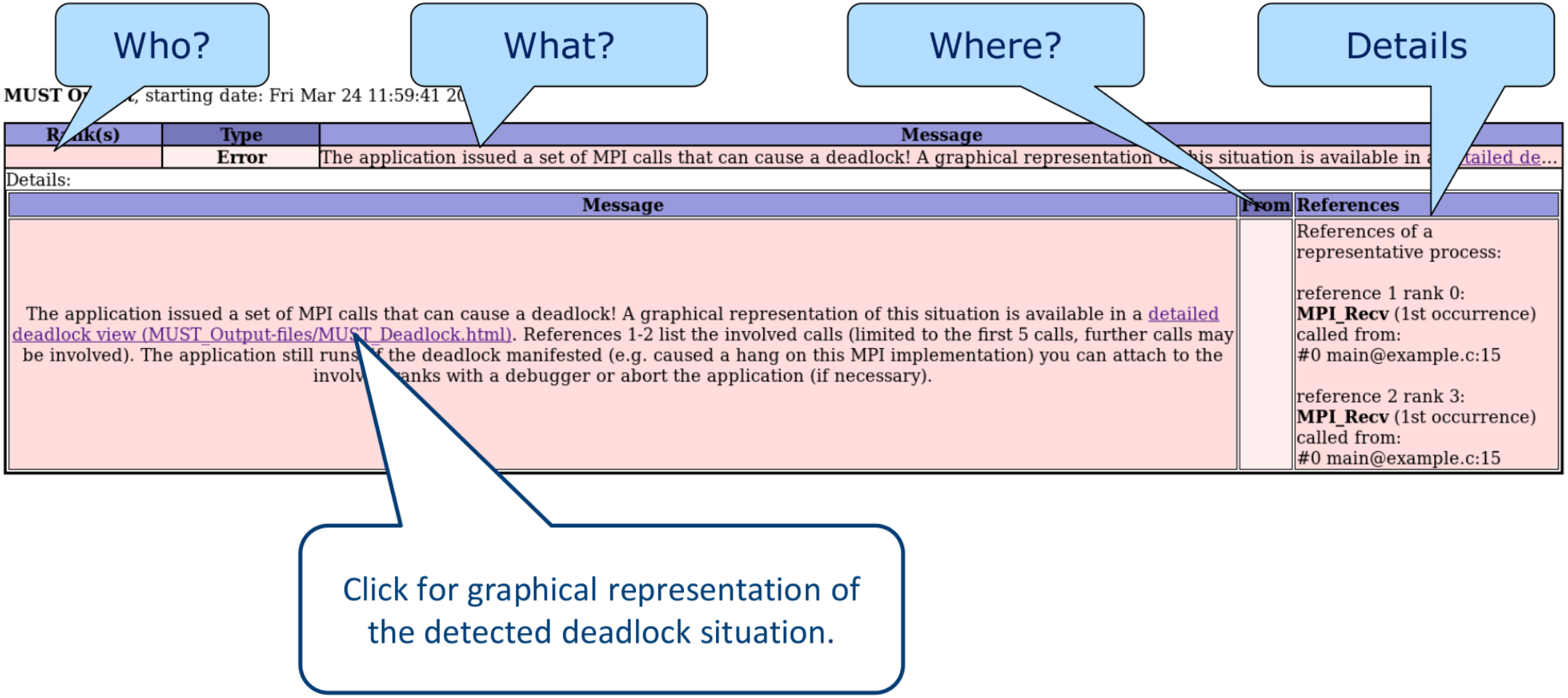
MUST EXAMPLE

```c
#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;
}
```

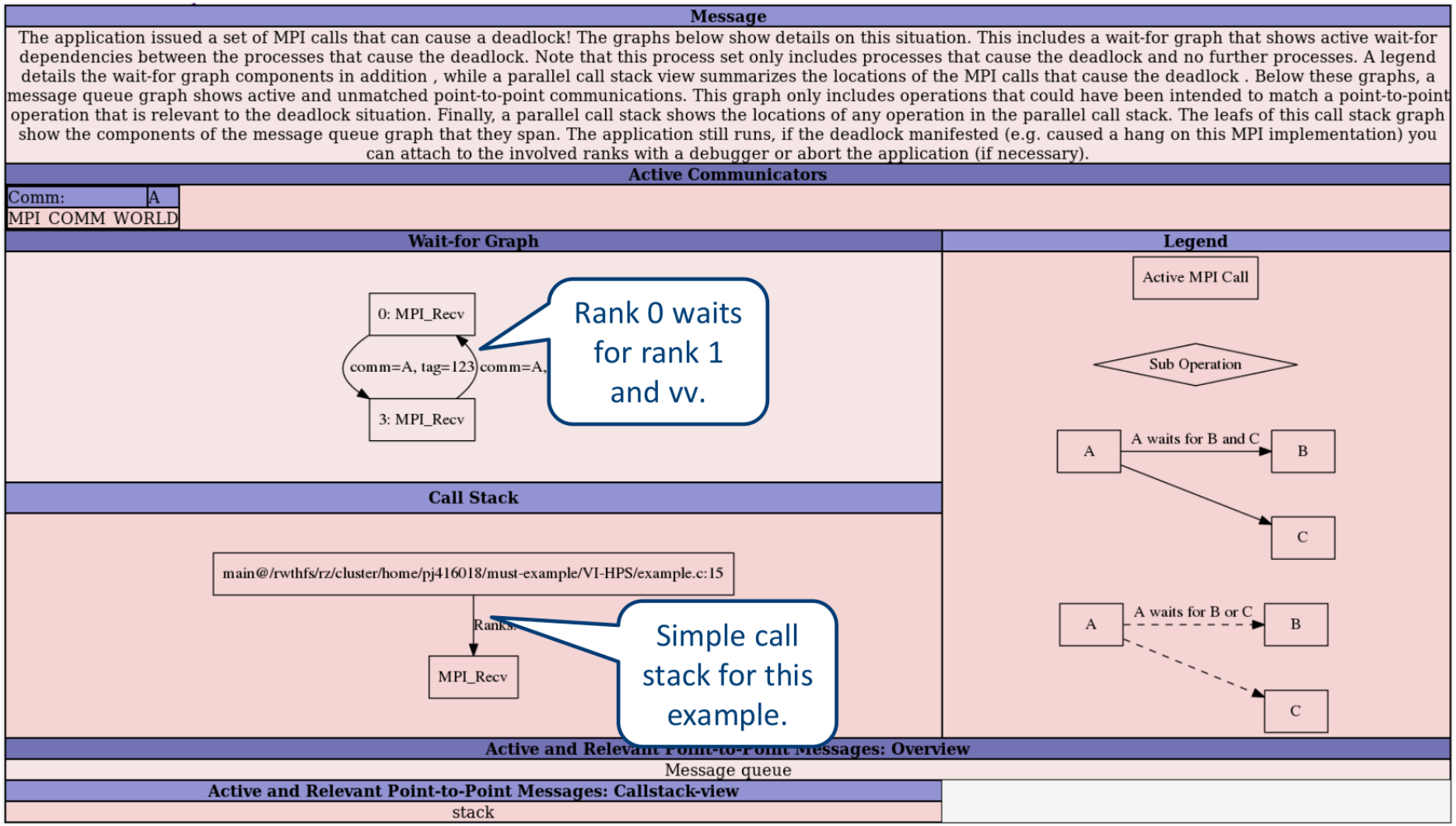
FIX 0: ADD MPI_INIT/MPI_FINALIZE

```c
#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



MUST DETECTS DEADLOCKS



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 … by 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

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 … receive operation!(In...
3(28795)	**Error**	The memory regions to be transfered by this send … receive operation!(In...
3(28795)	**Error**	A receive operation uses a (datatype,count) pair that can not … 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). 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

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...

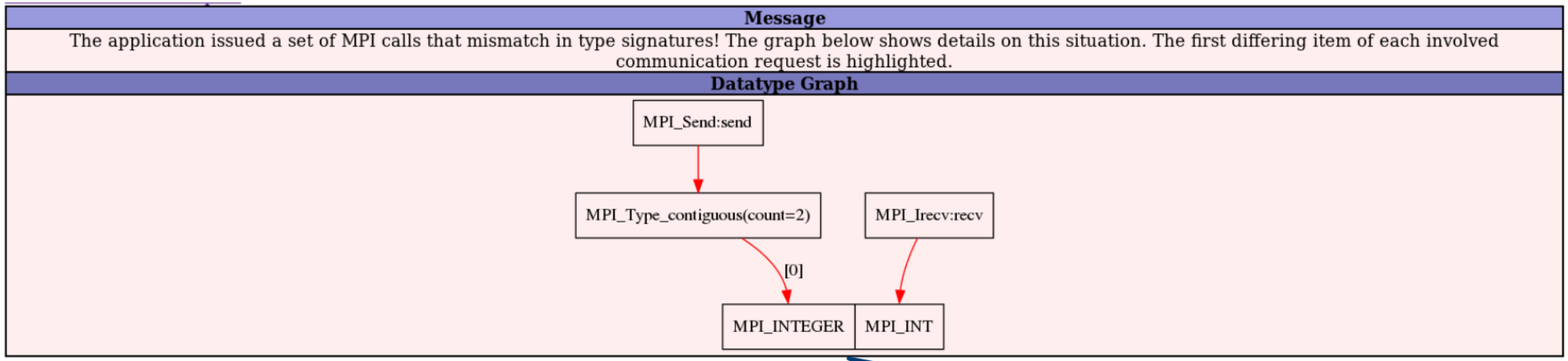
Use of Fortran type in C, datatype mismatch between sender and receiver

Use of uncommited datatype: **`type`**

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

MUST DETECTS DATATYPE ERRORS



Graphical representation of the type mismatch

FIX 3+4: C INT TYPE & USE TYPE_COMMIT

```c
#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…

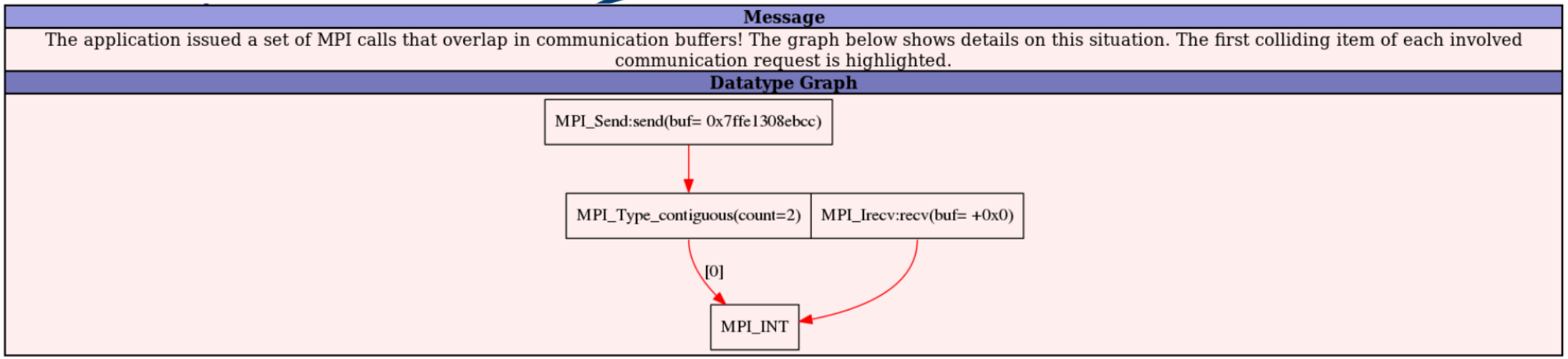
Details:

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



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 qua... pplication 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 locati... **MPI_Irecv** (1st occurrence, called from: #0 main@example-fix4.c:17	References of a representative ...rocess: re... ...1 rank 1: **MPI_Irecv** (1st occur... ...led from: #0 main@... ...x4.c:17

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

Unfinished non-blocking receive is resource leak and missing synchronization

Leak of user defined datatype object

FIX 6+7: USE MPI_WAIT & FREE DATATYPE

```c
#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

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

RUNNING MUST - CENTRALIZED

- **Slow, Centralized, Application may crash**
  - mustrun --np *X* exe
  - One additional analysis process
  - High overhead, < 32 processes
- **Fast, Centralized, Application may not crash**
  - mustrun --np *X* --must:nocrash exe
  - One extra analysis process
  - Limited scalability, < 100 processes
- **Fast, Centralized, Application may crash**
  - mustrun --np *X* --must:nodesize *Y* exe [--must:show]
  - $1 + [X/(Y-1)]$ analysis processes
  - Limited scalability, < 100 processes
  - Requires shared memory communication

RUNNING MUST - DISTRIBUTED

- **Distributed, Application may not crash**
  - mustrun --np *X* --must:distributed [--must:fanin *Z*] exe
  - Layer 0: $A = \lceil X/Z \rceil$
  - Layer 1: $B = \lceil A/Z \rceil$
  - …
  - Layer k: 1
  - Scalability > 10000 processes
- **Distributed, Application may crash**
  - mustrun --np *X* --must:distributed --must:nodesize *Y* [--must:fanin *Z*] exe
  - $A = \lceil X/(Y-1) \rceil \rightarrow B = \lceil A/Z \rceil \rightarrow C = \lceil B/Z \rceil \rightarrow \ldots \rightarrow 1$
  - Scalability ~ 5000 processes
  - Requires shared memory communication

QUESTIONS