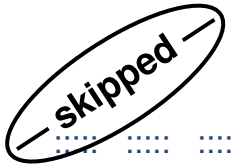
MPI-3.0 Fortran Draft

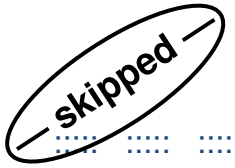
(MPI Forum Meeting July 2011)

The Fortran Working Group



Major Fortran Goals

- Guaranteed compile-time argument checking
- Better solutions for optimization problems
- Allowing any subarray everywhere, e.g., MPI_ISEND(a(1:100:5), …)
- Backward compatibility



Outline

- Goals
  - Why do we need the new functionality
  - Looking at the user survey
  - Conclusion on goals
- Answers from the Fortran WG5/J3 meeting, June 2011
  - Handles with BIND(C) versus SEQUENCE
  - Nonblocking and TARGET / ASYNCHRONOUS attribute
  - MPI_ALLOC_MEM
  - Conclusions on the WG5/J3 meeting results
- Technical questions and decisions
  - Do we really want that STATUS_IGNORE is substituted by function overloading
  - TR29113 interfaces and implication for PMPI
  - About handling callbacks
  - Support for Fortran Language Versions
  - Still unsupported features
- Final questions

Goals / New Features for New Fortran Interface

- Guaranteed compile-time argument checking
  - Unique MPI handle types
  - Explicit Interfaces
  - Instead of allowing implicit interfaces
- Fix non-blocking issues
  - And still keep useful Fortran optimizations
  - Guaranteed safe methods for nonblocking communication
  - Instead of today's "good luck" programming methods
- Fix subarray issues
  - Strided access through the Fortran subscript triplets, e.g., a(1:100:5) (instead of MPI's derived datatypes as the only method)
  - Now allowed in nonblocking, and split-collective routines
  - Instead of restricting it to blocking routines

Fortran TR29113, June 2011 meeting: WG5/J3 decided to enhance the ASYNCHRONOUS attribute for (MPI) communication.

Based on TR29113 TYPE(*), DIMENSION(..)

New Features / Goals, Continued

- Further enhancements
  - IERROR argument optional
  - STATUS output argument optional
  - MPI_ALLOC_MEM with standardized C_PTR pointers
  - New clear rules for using Fortran derived types as send/recv buffers
  - New solutions for code movement optimization problems:
    - Additional MPI routine: MPI_F_SYNC_REG
      Instead of user defined “DD” routine in MPI-2.0 – MPI-2.2
    - Using the **ASYNCHRONOUS** attribute
- Backward compatibility
  - Within mpif.h and mpi module:
    Only backward compatible enhancements
  - Further enhancements only within the new mpi_f08 module
  - Declaring mpif.h as “use is strongly discouraged”

We’ll rethink about and vote again

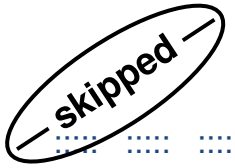
Does not need TR29113

TR29113 is required

Why do we need these new features?

- Many Fortran MPI applications are full of errors due to
  - Missing ierror argument
    - Implication: a zero may be written somewhere into the memory
  - Status may be declared wrong (not as an array)
    - Implication: some values may be written somewhere into the memory
- Huge development costs due to missing compile-time argument checking in mpif.h
- The Fortran strided arrays and subarrays (subscript triplets) cannot be used in nonblocking communication.
  - MPI_SUBARRAY datatype is so horrible that apps actually use copies (!)
- No portable usage of one-sided communication together with MPI_ALLOC_MEM
  - Required for portable use of MPI_LOCK routines
  - Also used in new interfaces in MPI-3.0 RMA
    - Implication: One has to use non-portable Cray pointers

WG5/J3 June 2011 meeting:
TR29113 enhances Fortran BIND(C) pointers to allow overloading existing Cray-pointer interface with a new standardized interface



Why do we need these new features, Continued

- Currently, usage of derived types as send/recv buffers is based on
  - "Use of the SEQUENCE attribute may help here, somewhat." (MPI-2.2 p484:42-43)
    - Implication: no portable use of derived type buffers
- There are significant problems between MPI nonblocking communication and Fortran compiler based optimization.
  - The current description is incomplete.
  - The proposed solutions are also incomplete.
    - Implication: The combination of incomplete description and proposed solutions is wrong
    - Implication: Applications may be programmed fully according MPI-2.2 and may still produce wrong results according to this Fortran-MPI-mismatch

User's Answers (based on the SC09 MPI Forum Questionnaire)

Question 16:

Rank the following in order of importance to your MPI applications (1=most important, .. 6=least important)

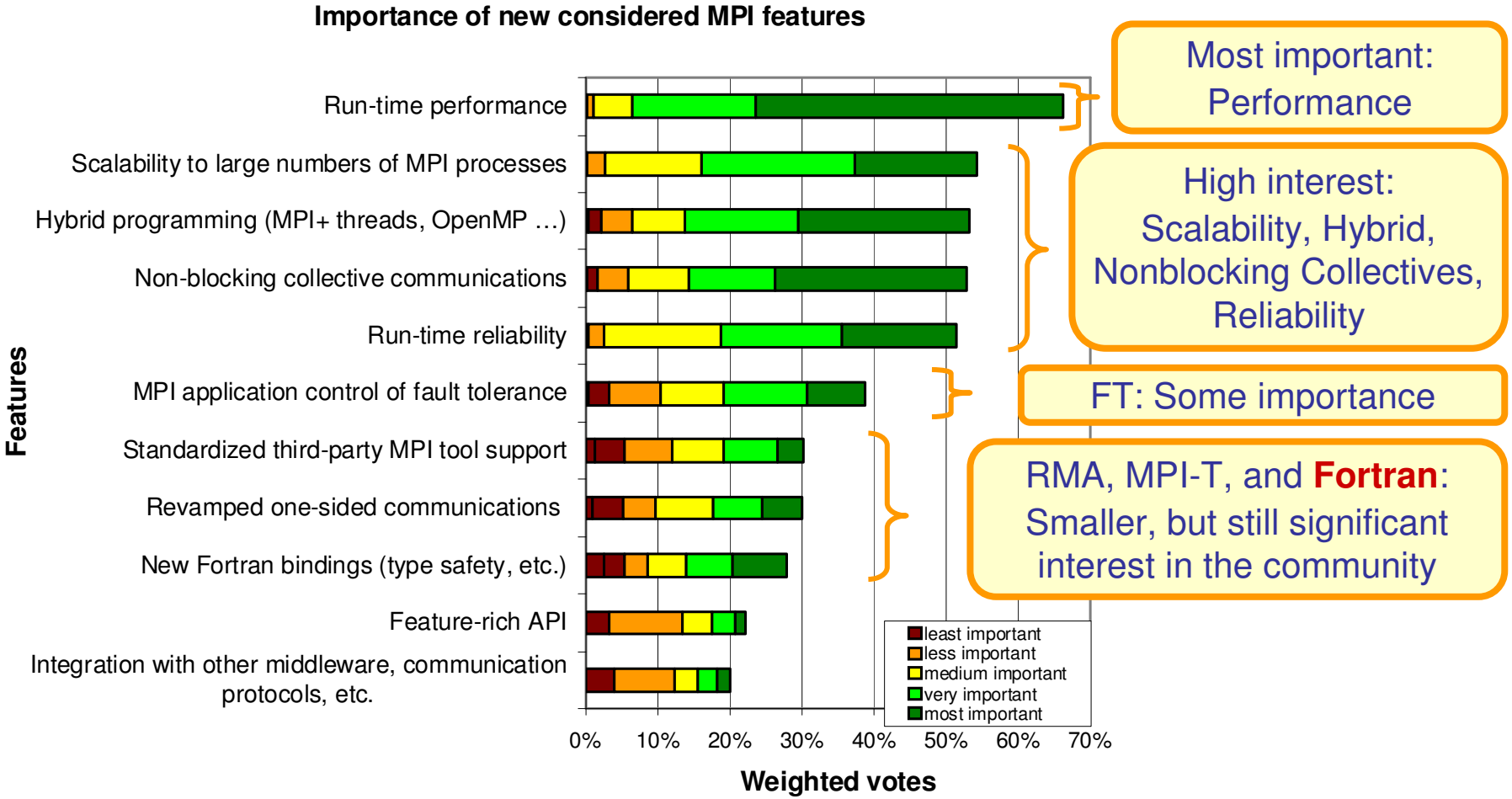
- **New Fortran bindings (type safety, etc.)**
  - 838 users answered Question 16
  - 628 users answered the Fortran line of Question 16:
    - **68 (=10.8%)** answered with 1 = **most important**
    - **72 (=11.5%)** answered with 2 = **very important**
    - **78 (=12.4%)** answered with 3 = **medium important**
    - 64 (=10.2%) & 99 (=15.8%) with 4 & 5 = less important
    - 247 (=39.3%) answered with 6 = least important (expected to be mostly C users)

  i.e., 35% of the 628 Fortran answers and 26% of all 838 Q16 answers are in the range medium ... most important (218 answers)

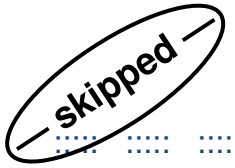
Question 2 showed that only 303 participants are "MPI application developer". We didn't asked about their programming language.

User's Answers in Relation to other Considered Features

(based on Question 16 and 17 of the questionnaire)



Weights: The importance range was set from 0 (=least) to 1 (most) with 6 and 5 (in Q16/17) equal-sized intervals. The interval-centers were used as weights. Vote percentage based on all 838 users.



Conclusions on our goals

- Guaranteed compile-time argument checking
  - Enhanced check for all different handles and integers
- Real solution for optimization problems with nonblocking routines
  - Correct rules to solve the MPI-Fortran-mismatch
- Allowing any subarray everywhere, e.g., MPI_ISEND(a(1:100:5), …)
  - As soon as Fortran 2008 + TR 29113 is available
- Source-code backward compatibility
  - For mpi.f and mpi module
- ABI backward compatibility

Consequences from the Fortran WG5/J3 meeting, June 2011

- Handles
  - Previously: we said there were restrictions on F08 handles
  - Now: Our BIND(C) based handles may need 8 bytes (**C compiler dependant**)
    - Compared to SEQUENCE based handles (4 bytes)
    - Or old style INTEGER handles (4 bytes)
  - They can still be used in COMMON blocks
  - Therefore no serious problem with changing applications from old mpi module to new mpi_f08 module
- Nonblocking
  - Previously: we solved this with the TARGET attribute
  - Now: ASYNCHRONOUS will work → we add it

New with TR29113

  - But we additional need (we write it into our requirements-section):
    - **"The Fortran compiler must guarantee that the ASYNCHRONOUS attribute protects Fortran asynchronous I/O and also MPI communication if the compiler provides TYPE(*), DIMENSION(..), i.e., the TR29113 must be implemented as a whole."**

skipped

Technical background slide

Answers from the Fortran WG5/J3 meeting, June 2011(details)

- New handles as derived types: With BIND(C) versus SEQUENCE
  - With SEQUENCE

```
TYPE :: MPI_Comm
    SEQUENCE
    INTEGER   :: MPI_VAL
END TYPE MPI_Comm
TYPE(MPI_Comm) :: array_of_communicators(5)
```

    provides the same data space as, i.e., usually **5x4 bytes**

```
INTEGER :: array_of_communicators(5)
```

  - With BIND(C)

```
TYPE, BIND(C) :: MPI_Comm
    SEQUENCE
    INTEGER   :: MPI_VAL
END TYPE MPI_Comm
TYPE(MPI_Comm) :: array_of_communicators(5)
```

    provides the same data space as and a array of such C structs, i.e., usually **5x8 bytes**

We still want to use BIND(C) because SEQUENCE does not work with the new BIND(C) interfaces

Answers from the Fortran WG5/J3 meeting, June 2011 (details)

- Buffers in nonblocking routines protected with the TARGET attribute
  - The proposal to use the TARGET attribute may be wrong.
  - TARGET may be also needed for the buffer dummy arguments.
  - Recommendation by the Fortran committee: Don't use TARGET.
- Protection with the ASYNCHRONOUS attribute
  - WG5/J3 decided to enhance the existing ASYNCHRONOUS attribute to protect also communication buffers.
  - WG5/J3 decided not to provide a method to check which semantics the ASYNCHRONOUS attribute has:
    - Old Fortran 2003/2008 without TR29113, i.e., protecting only Fortran asynchronous I/O
    - New TR29113 semantics, i.e., protecting asynchronous I/O & MPI
  - We should state in our requirement section:
    - **"The Fortran compiler must guarantee that the ASYNCHRONOUS attribute protects Fortran asynchronous I/O and also MPI communication if the compiler provides TYPE(*), DIMENSION(..), i.e., the TR29113 must be implemented as a whole."**

MPI_ALLOC_MEM and Fortran

I want to acknowledge the WG5/J3 committee for detecting and solving this problem.

- How to use MPI_ALLOC_MEM together with C-Pointers in Fortran. (instead of non-standard Cray-Pointers)

new

Also available in the mpi module, not in mpif.h

```
SUBROUTINE MPI_Alloc_mem(size, info, baseptr, ierror)
  USE, INTRINSIC :: ISO_C_BINDING
  INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: size
  TYPE(MPI_Info), INTENT(IN) :: info
  TYPE(C_PTR), INTENT(OUT) :: baseptr ! overloaded with the following…
  INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(OUT) :: baseptr ! …type
  INTEGER, OPTIONAL, INTENT(OUT) :: ierror
END
```

new

```
…ROUTINE MPI_Free_mem(base, ierror)
  TYPE(*), DIEMSION(..) :: base
  INTEGER, OPTIONAL, INTENT(OUT) :: ierror
END
```

- New Example 8.1:

new

```
USE mpi_f08   !  or USE mpi   (not guaranteed with INCLUDE 'mpif.h')
USE, INTRINSIC :: ISO_C_BINDING
TYPE(C_PTR) :: p
REAL, DIMENSION(:,:), POINTER :: a                ! no memory is allocated
INTEGER, DIMENSION(2) :: shape
INTEGER(KIND=MPI_ADDRESS_KIND) :: size
shape = (/100,100/)
size = 4 * shape(1) * shape(2)                    ! assuming 4 bytes per REAL
CALL MPI_Alloc_mem(size,MPI_INFO_NULL,p,ierr)     ! memory is allocated and
CALL C_F_POINTER(p, a, shape)                     ! now accessible through a
…
A(3,5) = 2.71;
…
CALL MPI_Free_mem(a, ierr)                        ! memory is freed
```

Currently, only allowed for Fortran types that are compatible with a type in C.

TR29113: This restriction is removed.

Thanks to Dieter an Mey, who gave me an example in Feb. 2004

Conclusions on the Fortran WG5/J3 meeting results

- It is a major step that WG5/J3 changed the **ASYNCHRONOUS** attribute to solve the MPI-Fortran nonblocking-problem.
  - This is really needed!
- It is bundled with the TR29113, which mainly defines **TYPE(*), DIMENSION(..) in combination with BIND(C) interfaces.**
- Especially for the nonblocking-problem, we would like to see efficient TR29113 compilers as soon as possible.
- This will happen only, if we include **TYPE(*), DIMENSION(..) & BIND(C)** as in integral part of MPI-3.0
- Therefore, a reasonable recommendation would be:
  - New mpi_f08 module uses BIND(C) interfaces with TYPE(*), DIMENSION(..) buffer arguments

- **With mpi_f08 requiring TR29113, we give an important force**
  - **To have TR29113 early implemented to allow new mpi_f08 module**
  - **To have with TR29113 correctness for nonblocking routines for all three Fortran support methods: mpif.h, mpi & mpi_f08 module.**

Technical issues

Still some open questions – decisions by the MPI Forum are needed

- Do we really want that STATUS_IGNORE is substituted by function overloading
  - Doubling the interfaces of 36 routines
    → tools have to profile all these symbols, too (36 x 2 = 72)

Technical Background slides

- TR29113 interfaces and implication for PMPI
  - Related to the tools interface
  - ABI interface: backward compatibility through different binding names
  - Long-term view: only one set of Fortran routines or wrappers

From a former meeting:

MPI_STATUS(ES)_IGNORE with function overloading

With USE mpi_f08, the user can freely choose

- CALL MPI_Recv(buf,cnt,datatype,src,tag,comm,status,ierror)
- CALL MPI_Recv(buf,cnt,datatype,src,tag,comm, ierror)
- CALL MPI_Recv(buf,cnt,datatype,src,tag,comm,status)
- CALL MPI_Recv(buf,cnt,datatype,src,tag,comm)

- Some routines are often in the critical path:
  - Function overloading is at compile-time
  - → no conditional branch at run-time
  - → Function overloading is more efficient

→ Same API cannot be done with OPTIONAL status argument, i.e., with OPTIONAL status, users must write

- CALL MPI_File_write(fh,buf,count,datatype, IERROR=ierror)

instead of

- CALL MPI_File_write(fh,buf,count,datatype, ierror)

- Also MPI_ERRCODES_IGNORE and MPI_UNWEIGHTED

new

Note that here, ierror may be needed, because in all I/O routines, ERRORS_RETURN is the default!

Same decisions as in C++

Straw Votes:	Yes	No	Abstain
	5	**2**	5

How to handle MPI_STATUS_IGNORE within mpi_f08?

Do we really want to substitute MPI_STATUS_IGNORE by such optional method (i.e., through function overloading)?

- Disadvantage:
  - The MPI implementers have to handle 2 routines
  - The tools people have also to handle both routines
  - The user has to be aware of the two binding names when he wants to write his/her own wrappers or pairs of two wrappers
- Advantage:
  - Natural for user and language
  - Can be faster (compile-time test instead of runtime branch)
- Disadvantage:
  - Most implementation may implement it through a wrapper, i.e., will pass internally again the MPI_STATUS_IGNORE
- Implications:
  - Only more complicated without a real win

```
INTERFACE MPI_Recv
  SUBROUTINE MPI_Recv_f08 (buf, …, status, ierror) &
  & BIND(C, NAME='MPI_Recv_f08')
    TYPE(*), DIMENSION(..) :: buf
    …
    TYPE(MPI_Status), INTENT(OUT) :: status
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
  END SUBROUTINE
  SUBROUTINE MPI_Recv_f08_nostatus (buf, …, ierror) &
  & BIND(C, NAME='MPI_Recv_f08_nostatus')
    TYPE(*), DIMENSION(..) :: buf
    …
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
  END SUBROUTINE
END INTERFACE
```

Old votes

Straw	Yes	No	Abstain
Votes:	5	**2**	5

New votes:

Straw	Yes	No	Abstain
Votes:	1	11	8

skipped

Technical background slide

How to define the TR29113 interfaces in mpi_f08 and implication for PMPI

```
INTERFACE MPI_Send
        SUBROUTINE MPI_Send_f08 (…) BIND(C,NAME='MPI_Send_f08')
```

→ For the profiling interface, a portable binding name is used: MPI_Send_f08

→ If a user wants to program a profiling wrapper, he/she must also define

```
        SUBROUTINE MPI_Send_f08 (…) BIND(C,NAME='MPI_Send_f08')
```

→ MPI implementers can directly use the advantages of TR29113

→ Requires that the Fortran standardization has allowed all MPI arguments through BIND(C)
→ e.g., callbacks must be done through the TR29113 public comment period.

With a preliminary implementation, the user has to define

```
        SUBROUTINE MPI_Send_f08 (buf, …)
          !$PRAGMA IGNORE_TKR buf
          REAL :: buf(*)
```

skipped

Technical background slide

Discussed with Tools Group (Martin Schulz): The at maximum 4 sets of wrappers/routines are okay as long as the routine names are systematic.

Choice buffers with TR29113 and ABI backward compatibility

- For the application, the Fortran routine name is always the same, e.g., **MPI_Recv**
- **mpi_f08 module:**
  - Final implementation with TYPE(*),DIMENSION(..) buffers & BIND(C) interfaces
    - Internal binding names: e.g., **MPI_Recv_f08** (~~and MPI_Recv_f08_nostatus~~)
  - Preliminary implementation with IGNORE_TKR buffers & without BIND(C)
    - Internal binding names: Mapped by the Fortran compiler, e.g., SUBROUTINE MPI_Recv_f08 → binding name **mpi_recv_f08__** ~~(and mpi_recv_f08_nostatus__)~~
- **mpi module and mpif.h:**
  - Old/current implementations: Choice buffers with IGNORE_TKR buffers & interfaces without BIND(C) **(or implicit interface in mpif.h)**
    - Internal binding names: Mapped by the Fortran compiler, e.g., SUBROUTINE MPI_Recv → binding name **mpi_recv__**
  - Future implementation with TYPE(*),DIMENSION(..) buffers & BIND(C)
    - Internal binding names: e.g., **MPI_Recv_f** (nostatus via MPI_STATUS_IGNORE)
- When switching to TR29113 implementation, one must keep preliminary/old routines in the library until all old application.o files (i.e, compiled with old module) are vanished

skipped

Technical background slide

Long-term implementer's view

- With BIND(C) interfaces, all three Fortran support methods
  - mpi_f08, mpi module, and mpif.h

  most routines can be implemented with **same** C-wrappers or C-routines.
- Exceptions:
  - ~~Routines with the optional arguments status, errcodes, weights~~
  - Routines with arrays of handles
  - (Routines with choice buffer
    if mpi.h does **not** use interfaces although Fortran 2008 + TR 29113 is available)
- These C-routines are different to the C MPI-bindings.
- Alternative: **A set of portable wrappers in Fortran** (also with all three support methods)
  - with a tiny C-layer to handle the portable TR 29113 dope vector
  - and calling the original C MPI-bindings.
- Performance:
  - Same bandwidth as with current compiler based triple copying
  - New optimization opportunity if combing dope vector and datatype handling

MPI_Recv_f08 and aliased MPI_Recv_f

Callback routine interfaces

- For callback routines with choice buffers:
  - MPI_User_function for MPI_Op_create
  - MPI_Datarep_conversion_function for MPI_Register_datarep
  - We keep the implicit interface (i.e., with EXTERNAL)
  - Because otherwise, the user has to handle the TYPE(*), DIMENSION(..)
- For all other callbacks, we defined ABSTRACT INTERFACEs
  - Implications:
    - The user written callback routines must have an explicit interface.
    - They must be part of a module or can defined locally.
    - Correctness of the argument list is always checked.

Support for Fortran Language Versions

- Full support for Fortran 2008 + TR 29113
- Nearly full support for Fortran 95 plus Fortran 2003 features
  BIND(C) + ISO_C_BINDING + PROCEDURE & ABSTRACT INTERFACE
  - Restrictions:
    - Subscript triplet subarrays cannot be used in nonblocking routines
    - New MPI_ALLOC_MEM interface can be used only for Fortran types that have an equivalent in C
  - All other features available in mpi_f08, mpi modules and mpif.h
- Support for pure Fortran 90 and 95
  - Restrictions:
    - Restricted support through modified internal representation of handles and statuses (SEQUENCE instead of BIND(C))
    - MPI_ALLOC_MEM restricted to MPI-2.0 – MPI-2.2 quality (i.e., Cray pointers)
    - Callback prototypes are implemented through internal INTERFACE, substituting PROCEDURE + ABSTRACT INTERFACE
- Restricted support for Fortran 77 was and is outside the scope of MPI-2.0 and later

Still unsupported features

- These problems can be solved in a future ticket / standardization
  - Overlapping of nonblocking communication and computation on different parts of the same data structure or array
  - (currently, I know only this problem)

Final question

- Based on text and discussion from last meeting and current decisions: When we write all down, which vote will you give for the whole Fortran package?

Straw Votes:	Yes	No	Abstain
	12	0	8

5 of them have not been at the May 2011 meeting, i.e., they have not been familiar with the text presented their. Therefore, they have too little information for voting.