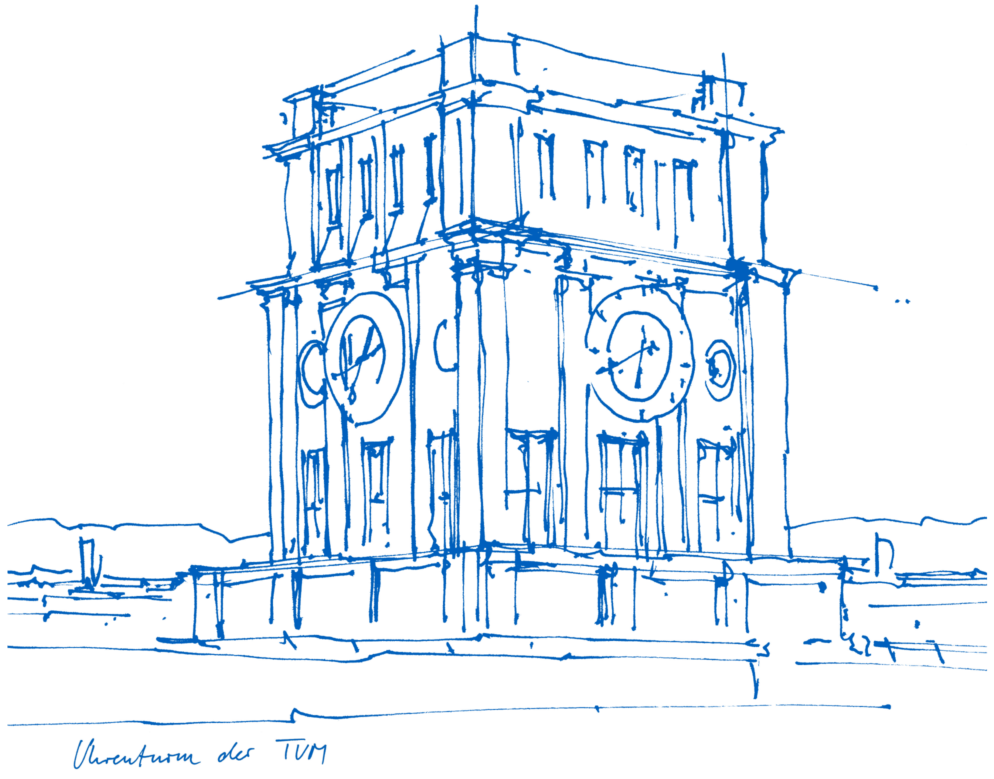
The Final Steps to MPI 4.0 …

… and What's Next?

Martin Schulz
Technische Universität München
Chair of the MPI Forum

+ the entire MPI Forum

EuroMPI 2020, Anywhere On Earth
(and virtually in Austin, TX)
September 2020



The MPI Forum Drives MPI

Standardization body for MPI

- Discusses additions and new directions
- Oversees the correctness and quality of the standard
- Represents MPI to the community

Organization consists of chair, secretary, editor, convener, and member organizations

Open membership

- Any organization is welcome to participate
- Consists of working groups and the actual MPI forum (plenary)
- ~~Physical~~ meetings 4 times each year (3 in the US, one with EuroMPI/Asia/USA)
  - Working groups meet between forum meetings (via phone)
  - Plenary/full forum work is done mostly at the ~~physical~~ meetings
- Voting rights depend on attendance
  - An organization has to be present two out of the last three meetings (incl. the current one) to be eligible to vote
- New items receive a “reading” and “2 votes”

The Bulk of Work is in the Working Groups

Collective Communication, Topology, Communicators, Groups
- Torsten Hoefler, Andrew Lumsdaine and Anthony Skjellum

Fault Tolerance
- Wesley Bland, Aurélien Bouteiller and Rich Graham

HW Topologies
- Guillaume Mercier

Hybrid and Accelerator Programming
- Pavan Balaji and Jim Dinan

Large Count
- Jeff Hammond and Anthony Skjellum

Persistence
- Anthony Skjellum

Point to Point Communication
- Rich Graham and Dan Holmes

Remote Memory Access
- Bill Gropp and Rajeev Thakur

Semantic Terms
- Rolf Rabenseifner and Purushotham Bangalore

Sessions
- Dan Holmes

Tools
- Kathryn Mohror and Marc-Andre Hermanns

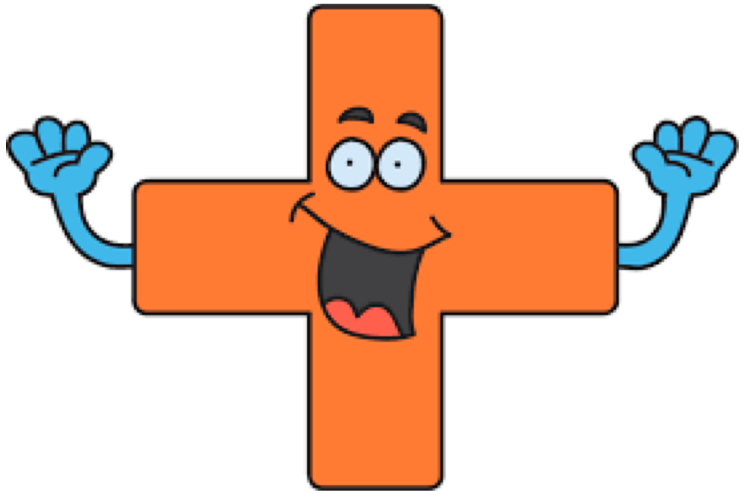
The Status of MPI

MPI 3.0 ratified in September 2012
- Major new functions

MPI 3.1 ratified in June 2015
- Minor updates and additions

Fully adopted in all major MPIs

MPI 4.0 work coming to an end
- Final discussions next week
- Hopefully Release Candidate for SC
- Will be available at http://www.mpi-forum.org/



Major additions for MPI 4.0
- Solution for “Big Count” operations
- Persistent Collectives
- Partitioned Communication
- Topology Solutions
- New init options via MPI Sessions
- Simple fault handling to enable fault tolerance solutions
- New tool interface for events
- Reworking of the terms chapter

Big Count

Problem: in current interface “count” arguments are “int”

- Limits communication volumes to 32bit x Datatype
- Significant number of applications need more
- Initial datatype “trick” no longer sufficient

Solutions discussed include:

- Just changing “int” arguments to “MPI_Count” arguments → ☹ ☹ ☹
- Polymorphic bindings → ☹ ☹
- Duplication of interfaces: with int and with MPI_Count (“_c” suffix) → ☹

Changes required

- Update of the general type rules for bindings
- Verification of all bindings, which will led to errata tickets
- Addition to “many” new routines with ”_c”

Status: minor fixes needed, to be decided next week

Big Count = Big Task

List of functions affected (133)

MPI_Accumulate
MPI_Allgather
MPI_Allgatherv
MPI_Allreduce
MPI_Alltoall
MPI_Alltoallv
MPI_Alltoallw
MPI_Bcast
MPI_Bsend
MPI_Bsend_init
MPI_CONVERSION_FN_NULL
MPI_Comm_spawn_multiple
MPI_Dist_graph_neighbors_count
MPI_Exscan
MPI_File_iread
MPI_File_iread_all
MPI_File_iread_at
MPI_File_iread_at_all
MPI_File_iread_shared
MPI_File_iwrite
MPI_File_iwrite_all
MPI_File_iwrite_at
MPI_File_iwrite_at_all
MPI_File_iwrite_shared
MPI_File_read
MPI_File_read_all
MPI_File_read_all_begin

MPI_File_read_at
MPI_File_read_at_all
MPI_File_read_at_all_begin
MPI_File_read_ordered
MPI_File_read_ordered_begin
MPI_File_read_shared
MPI_File_write
MPI_File_write_all
MPI_File_write_all_begin
MPI_File_write_at
MPI_File_write_at_a
MPI_File_write_at_all_beg
MPI_File_write_ordered
MPI_File_write_ordered_begin
MPI_File_write_shared
MPI_Gather
MPI_Gatherv
MPI_Get
MPI_Get_accumulate
MPI_Get_count
MPI_Get_elements
MPI_Get_elements_x
MPI_Graph_neighbors_count
MPI_Iallgather
MPI_Iallgatherv
MPI_Iallreduce
MPI_Ialltoall

MPI_Ialltoallv
MPI_Ialltoallw
MPI_Ibcast
MPI_Ibsend
MPI_Iexscan
MPI_Igather
MPI_Igatherv
MPI_Imrec
MPI_Neig allg
MP
M I_Iredu
MPI_Iredu scatter
MPI_Ireduce_scatter_block
MPI_Irsend
MPI_Iscan
MPI_Iscatter
MPI_Iscatterv
MPI_Isend
MPI_Issend
MPI_Mrecv
MPI_Neighbor_allgather
MPI_Neighbor_allgatherv
MPI_Neighbor_alltoall

MPI_Neighbor_alltoallv
MPI_Neighbor_alltoallw
MPI_Pack
MPI_Pack_external
MPI_Pack_external_size
MPI_Pack_size
MPI_Put
MPI_Raccumulate
MPI_R cv
Recv_init
uce_local
Reduce_scatter
Reduce_scatter_block
MPI_ et
MPI_Rget_accumulate
MPI_Rput
MPI_Rsend
MPI_Rsend_init
MPI_Scan
MPI_Scatter
MPI_Scatterv
MPI_Send
MPI_Send_init
MPI_Sendrecv
MPI_Sendrecv_replace
MPI_Ssend

MPI_Ssend_init
MPI_Startall
MPI_Status_set_elements
MPI_Status_set_elements_x
MPI_T_cvar_handle_alloc
MPI_T_pvar_handle_alloc
MPI_Testall
MPI_Testany
MPI_Testsome
MPI_Type_contiguous
MPI_Type_create_hindexed
MPI_Type_create_hindexed_block
MPI_Type_create_hvector
MPI_Type_create_indexed_block
MPI_Type_create_struct
MPI_Type_get_extent_x
MPI_Type_get_true_extent_x
MPI_Type_indexed
MPI_Type_size_x
MPI_Type_vector
MPI_Unpack
MPI_Unpack_external
MPI_Waitall
MPI_Waitany
MPI_Waitsome

We already have
a few "_X" functions

Pythonization

All bindings are generated via embedded Python

- Initially introduced as in a neutral way
- Vetted by all WGs
- Then used for BigCount „flipped a switch“

Consequences

- Slightly different rendering
- More consistency
- Uncovered errors

```
\begin{mpi-binding}
    function_name("MPI_Send")

    parameter(
        "buf", "BUFFER", desc="initial address of
         send buffer", constant=True)
    parameter(
        "count",
        "XFER_NUM_ELEM_NNI_SMALL",
        desc="number of elements in send buffer",)
    parameter(
        "datatype", "DATATYPE", desc="datatype of
         each send buffer element)
    parameter("dest", "RANK", desc="rank of
              destination")
    parameter("tag", "TAG", desc="message tag")
    parameter("comm", "COMMUNICATOR")
\end{mpi-binding}
```

Why is this important?

- Machine readable description of all MPI routines
- Eases future standard-wide changes
- Enables better tool support (e.g., generation of PMPI tools)

Status: integrated into the Standard’s Latex Source Code

Persistent Collectives

Following the basic ideas of persistent point to point

- One time initialization to pass all arguments, which returns a request
- Use of this request to start communication
- Completion using Test/Wait
- Reuse request to restart the operation as often as one wants

Available for all MPI collective communication operations (and barriers)

Why?

- Specify repeated operations
- Ability to lock down resources and to cache execution plan
- Performance optimization after (small) 1x cost
- Allows for continuous plan optimization

Status: voted into MPI 4.0

Partitioned Communication

Core idea

- Built on the concept of persistent operations
- Send buffers are split into partitions
  - Fill each partition and mark it as ready (from independent threads)
- Receive buffers are split into partitions
  - Individual notifications for each arriving partition
- Enables partial data transfers

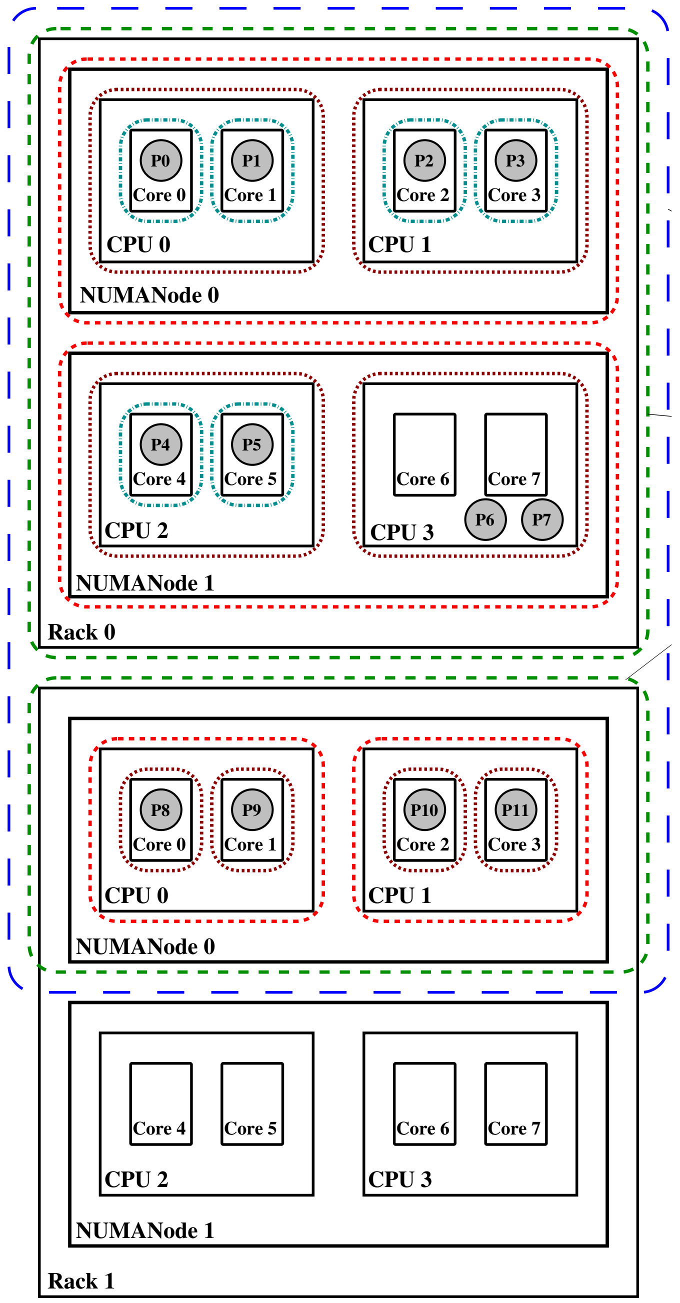
Notifications - on send and receive side – are light-weight

- May be driven from accelerators
- May need additional synchronization to trigger message transfer safely

So far only simple point-to-point options, more to come

Status: voted into MPI 4.0 (as a new chapter)

New Ways to Adapt to Topologies



New systems are hierarchical

- Mapping is critcal
- Need topology-aware communicators

Guided Mode

- MPI_COMM_SPLIT_TYPE
- Special split type
- Info key to specify level

Unguided Mode

- Start at WORLD
- Step-wise go to lower levels
- Iterative until leaf is reached

Query function missing

Status: voted into MPI 4.0

A New Way to Use MPI: MPI Sessions

Basic scheme

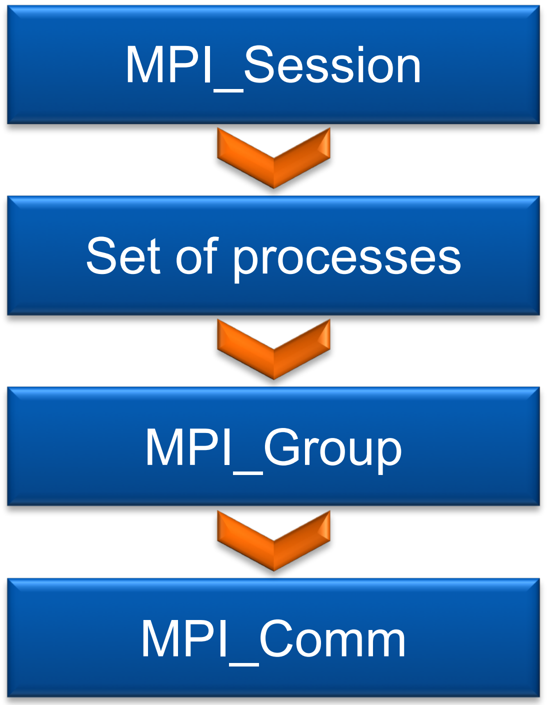
1. Get local access to the MPI library

   Get a Session Handle
2. Query the underlying run-time system

   Get a “set” of processes
3. Determine the processes you want

   Create an MPI_Group
4. Create a communicator with just those processes

   Create an MPI_Comm



MPI Session’s intended goals

- No more implicit MPI_COMM_WORLD
- Enable runtime information to flow into MPI
- Creation of communicators without parent communicators
- Resource isolation
- Many future uses … - more later

Status: voted into MPI 4.0

Improved Error Handling

Goal: allow applications to limit impact of failures to avoid terminations

- Specify that MPI_SUCCESS indicates only the result of the operation, not the state of the MPI library.
- Localize error impact of some MPI operations.
  MPI_ALLOC_MEM will now raise an error on COMM_SELF, not COMM_WORLD
- Specify that MPI should avoid fatal errors when the user doesn't use MPI_ERRORS_ARE_FATAL
- New MPI Error Handler - MPI_ERRORS_ABORT
- Allow the user to specify the default error handler at `mpiexec` time.

What can you do with this?

- Point to Point communication with sockets-like error handling
- Enables master/worker and other non-traditional types of applications
- Enterprise applications that want to move from sockets to MPI can do so.

Status: voted into MPI 4.0

MPI_T Events: Callback-driven event information

Motivation

- PMPI does not provide access to MPI internal state information
- MPI_T performance variables only show aggregated information

New interface to query available runtime event types

- Follows the MPI_T variable approach
- No specific event types mandated
- Event structure can be inferred at runtime

Register callback functions to be called by the MPI runtime

- Runtime may defer callback invocation (tool can query event time)
- Runtime may reduce restrictions on callback functions per invocation
- Callback can query event information individually or copy data en bulk

Status: voted into MPI 4.0

Other Additions

Assertions for message traffic to guide optimization

- Can state that an application doesn't use wildcards
- Enables traffic optimizations
- Great opportunities for implementations to optimize

Remove info key propagation on communicator duplication

- New function: MPI_Comm_idup_with_info
- Better control over properties attached to communicator

Clarification of what it means to query the info object attached to an MPI object

Deprecation of send cancel

- Long overdue ☺

Small fixes to the MPI Tools Information Interface

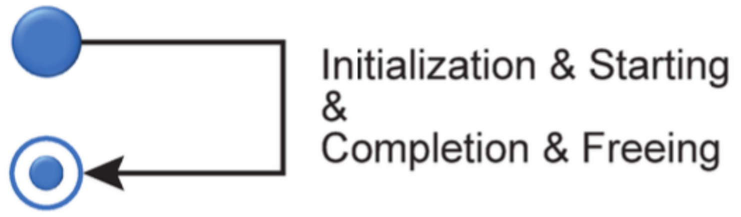
Access to MPI Info before MPI initialization (needed for Sessions, MPI_T, FT, …)

Update for the Terms Chapter

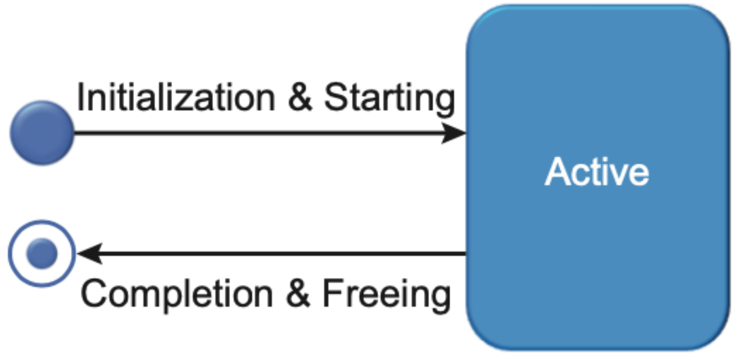
Terms Chapter needed updating to integrate new MPI APIs

- Led to a rewrite of a good part of them
- Helped to remove missunderstandings

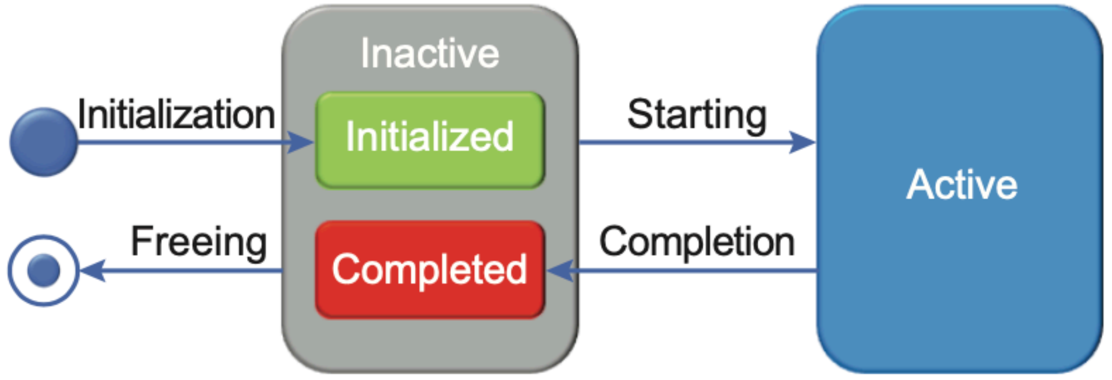
Type 1: Blocking operation



Type 2: Non-blocking operations



Type 3: Persistent operations



Status: up for 2nd vote, but RMA currently missing

Important Updates in the Terms Chapter

Local vs. Non-Local

Non-local procedure: An MPI procedure is non-local if returning may require, during its execution, some specific semantically-related MPI procedure to be called on another MPI process.

Local procedure: An MPI procedure is local if it is not non-local.

Incomplete / Blocking / Non-Blocking

Incomplete procedure: An MPI procedure is called incomplete if it is not a completing procedure. (see diagrams before – used to be called **non-blocking**)

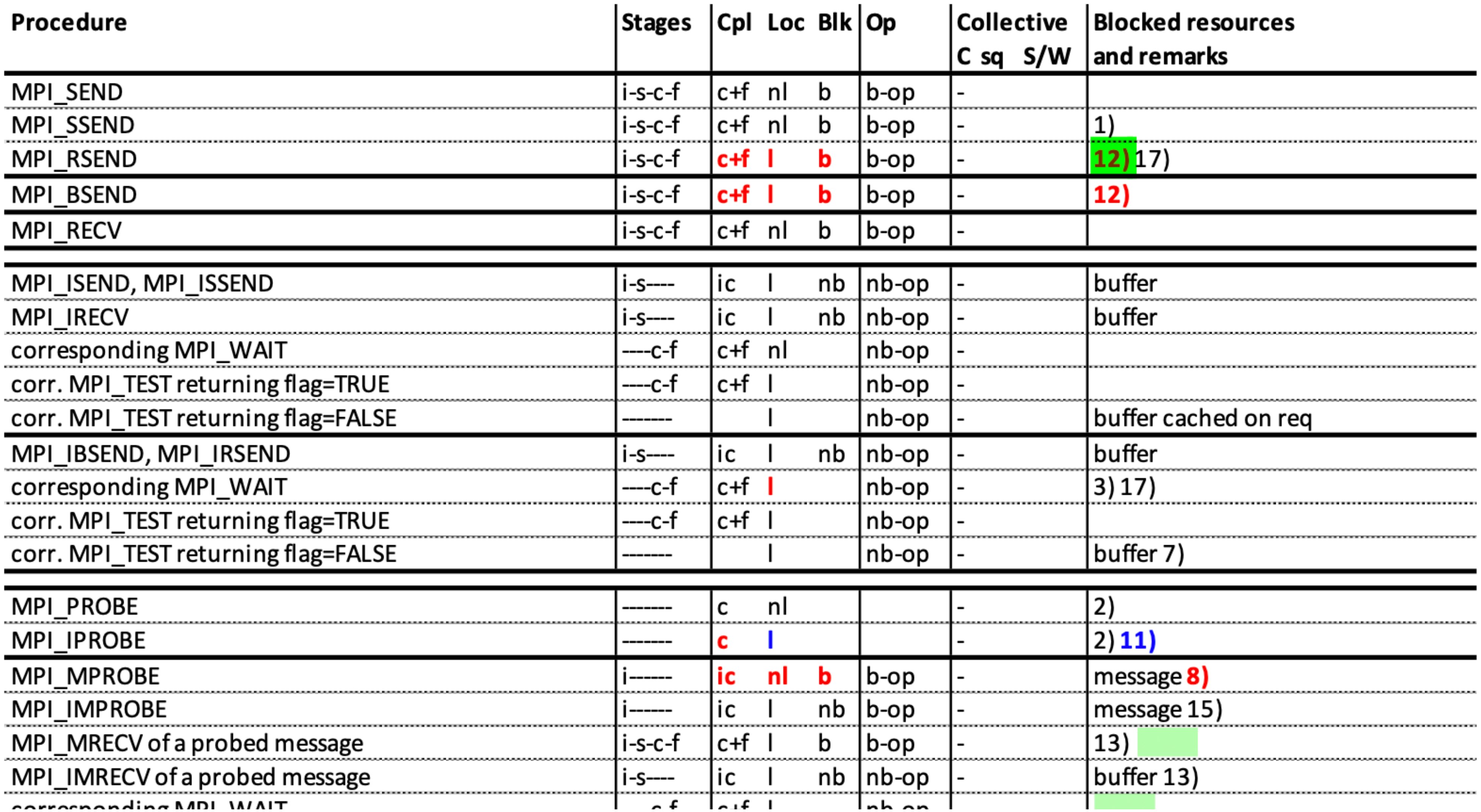
Nonblocking procedure: An MPI procedure is nonblocking if it is incomplete and local.

Blocking procedure: An MPI procedure is blocking if it is not nonblocking.

„The Table“ – A New Side Document

Classification of all MPI Routines

- Initially intended as an appendix
- Will be moved to a side document (for now)



Procedure	Stages	Cpl	Loc	Blk	Op	Collective	Blocked resources
						C sq S/W	and remarks
MPI_SEND	i-s-c-f	c+f	nl	b	b-op	-	
MPI_SSEND	i-s-c-f	c+f	nl	b	b-op	-	1)
MPI_RSEND	i-s-c-f	**c+f**	**l**	**b**	b-op	-	**12)** 17)
MPI_BSEND	i-s-c-f	**c+f**	**l**	**b**	b-op	-	**12)**
MPI_RECV	i-s-c-f	c+f	nl	b	b-op	-	
MPI_ISEND, MPI_ISSEND	i-s----	ic	l	nb	nb-op	-	buffer
MPI_IRECV	i-s----	ic	l	nb	nb-op	-	buffer
corresponding MPI_WAIT	----c-f	c+f	nl		nb-op	-	
corr. MPI_TEST returning flag=TRUE	----c-f	c+f	l		nb-op	-	
corr. MPI_TEST returning flag=FALSE	-------		l		nb-op	-	buffer cached on req
MPI_IBSEND, MPI_IRSEND	i-s----	ic	l	nb	nb-op	-	buffer
corresponding MPI_WAIT	----c-f	c+f	**l**		nb-op	-	3) 17)
corr. MPI_TEST returning flag=TRUE	----c-f	c+f	l		nb-op	-	
corr. MPI_TEST returning flag=FALSE	-------		l		nb-op	-	buffer 7)
MPI_PROBE	-------	c	nl			-	2)
MPI_IPROBE	-------	**c**	**l**			-	2) **11)**
MPI_MPROBE	i------	**ic**	**nl**	**b**	b-op	-	message **8)**
MPI_IMPROBE	i------	ic	l	nb	b-op	-	message 15)
MPI_MRECV of a probed message	i-s-c-f	c+f	l	b	b-op	-	13) 🟩
MPI_IMRECV of a probed message	i-s----	ic	l	nb	nb-op	-	buffer 13)
corresponding MPI_WAIT	----c-f	c+f	l		nb-op		🟩

The Final Steps to MPI 4.0

Current plan of record

- Final deliberations next week at the MPI Forum
  - Requires a few “No-No“ votes for final changes

- Declare Release Candidate (RC) document
- RC will serve as the „annual report“
  - Feedback from the wider community

- MPI-Forum in December: Validation and First vote
- MPI-Forum in February: Second vote and final ratification

Timeline may slip, we will know next week

WHAT NEXT?

MPI 4.1: A Clean-Up Pass for the MPI Standard?

No semantic changes, but …

Continued update of new Terms and Conventions

- One-sided communication
- Integration of “The Table”

Terminology cleanup

- Process vs. MPI Process
- MPI Sessions vs. MPI_T Sessions

Consistent bindings

Chapter re-organization

Editorial changes and fixes

“Trivial“ Additions

Fill in routines to preserve orthogonality
- More non-blocking routines
- More persistent routines

Expand concept of partitioned communication
- More complex scatter/gather
- Partitioned collectives (perhaps not that trivial ☺)

More info keys to communicate application intent

Fortran bindings for MPI_T
- Prepare for a more general usage of CVARs

Mandatory MPI_T PVARs, CVARs, and Events

And then? A View Through the Crystal Ball

Adaptivity (in MPI and elsewhere) will be Key

Adaptivity is needed for efficient resource utilization

- Worst case provisioning is a waste of resources
- But limited resources lead to contention, variability, …
- Need to actively manage resources (e.g., for power, I/O, …)

Adaptivity is needed to counteract variability

- Increasing variability will be the new normal
- Need to actively balance and shift workloads

Adaptivity is needed to manage complex workflows

- Single, static applications in pure SPMD style will be a thing of the past
- Coupling of components for UQ and/or scale bridging
- Need to actively schedule components with varying demands

Adaptivity is needed by new workloads, especially in ML/DL/AI

BUT: must be coupled with adaptive runtime resource management:
in the hardware, middleware and the application – and that includes MPI

Towards a Global Session (Bubbles) Concept

MPI Sessions provide the ability to avoid global operations, use subsets
- Resource reduction and increased scalability
- But: as they are local, program subsets are hard to build on it

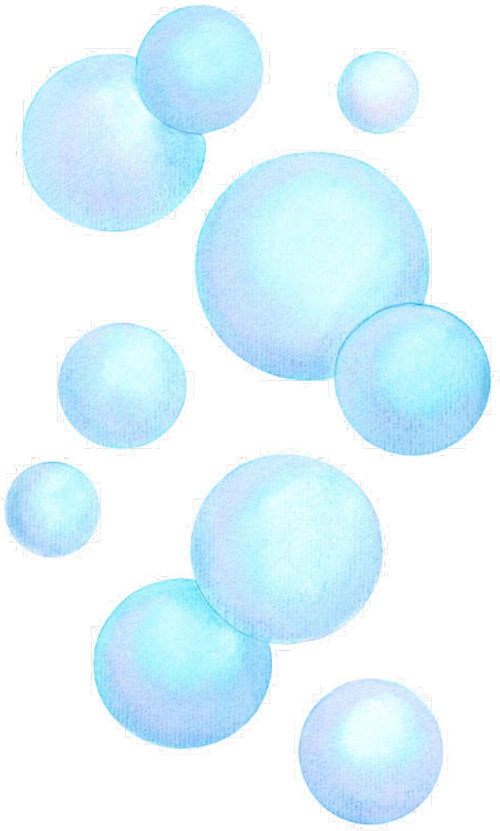
Ability to reason about all MPI objects that are
- … derived from the same local local session(s)
- … part of the same process groups

These objects form a natural group: an **"MPI bubble"**
- Isolated from the world in terms of communication
  - Within a bubble: "Normal MPI"
- Could be revoked/popped without global impact
- Granularity of malleability / components

What is missing? Variability in the process sets
- Hard: Dynamically changing number of process sets
- Harder: Dynamically changing sets
- Hardest: Dynamically changing communicators

Also: addressability and external connectivity of MPI Bubbles

Other Usage Scenarios for Sessions



Fault isolation

- Failures bring down bubbles
- Remaining bubbles can continue

Better support for hybrid execution

- See talk at EuroMPI19 by Jean-Baptiste Besnard

Workflow support

- Bubbles as components
- Separation of compute and visualization and/or tools

And much more …

An Updated PMPI: QMPI

Tools have become much more diverse

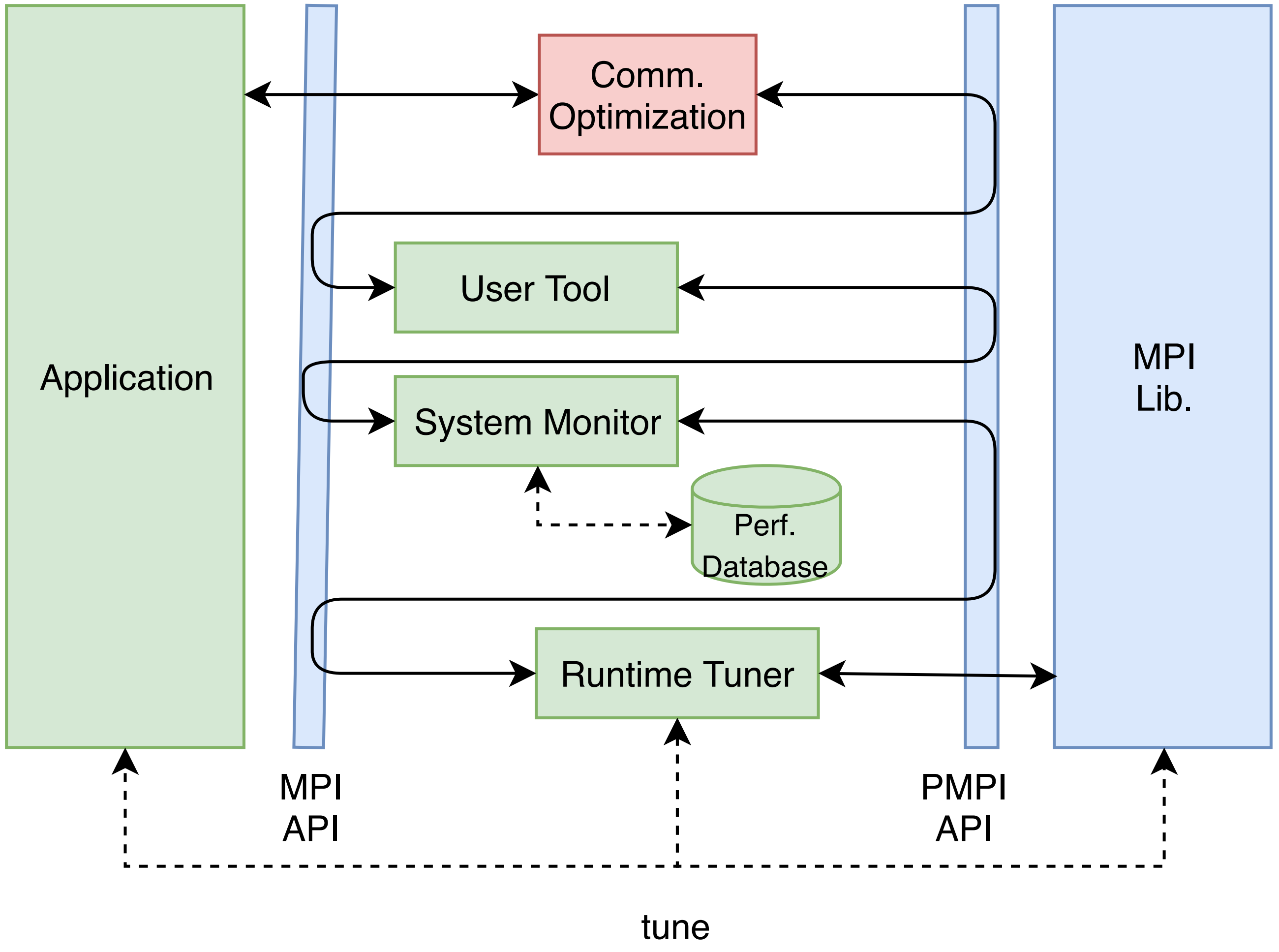
- From single profilers …
  … to wide range of concurrently used system, runtime and user tools

Driven by different „users“

- End-user
- App developer
- MPI developer
- Administrator

QMPI as one option

- Tools as dynamic wrappers
- Language independent
- Tool chains
- See talk at EuroMPI19 by Bengisu Elis



Separation of Bindings

New user communities use new/other languages

- C++, Python, Java, …
- MPI should be available to them, in their „native usage pattern“, not as C wrappers

One idea:

- Split the MPI standard into semantics and bindings
- One central semantics document
- One document per language to describe the mapping

Side effect

- Would make us think more about the details of the semantics of the standard

Question: what does this mean for scripted languages?

Other Ideas

MPI Subsetting

Support for Smart-Networks
- Processing on the NIC and Switches
- Need ability to express on the fly computation

MPI Schedules
- Predefined, persistent set of operations

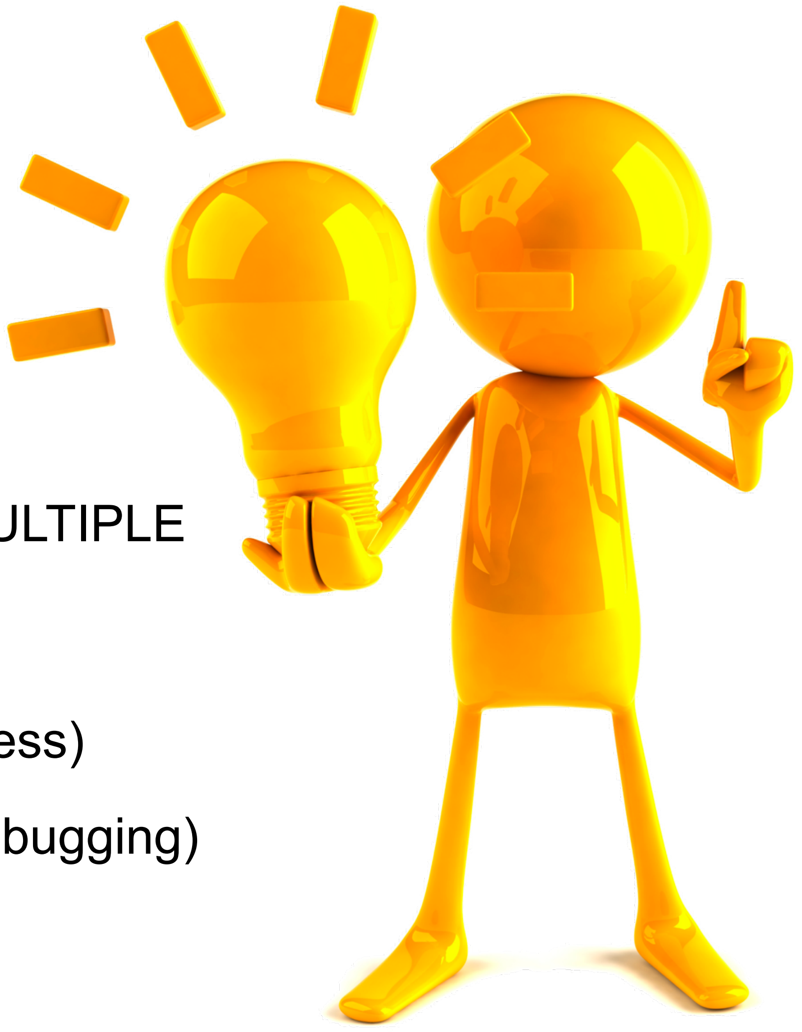
New thread levels “just below” MPI_TRHEAD_MULTIPLE

Better integration with task models
- Task activation or active messages
- Exposure or integration of task engines (progress)

Better exposure of runtime interfaces (e.g., for debugging)

Full featured FT support (with several modi)

Fixed release schedule



Participate!!!

Many exciting avenues to work on MPI

- Improving support for classic HPC applications
- Opening up new usage modes and communities
- Adding modern software design principles

The MPI Forum is always looking for new participation!

- Work on MPI 4.0 is (almost) done – feedback wanted
- Work on MPI 4.1 ramping up / focus on cleaning up the standard
- MPI 5.0 open for new ideas / open playground!

Get involved

- Let us know what you or your applications need
  - mpi-comments@mpi-forum.org
- Participate in WGs
  - Email list and Phone meetings
  - Each WG has its own Wiki
- Join us at a MPI Forum F2F meeting
  - Meetings are ***virtual*** for the time being

http://www.mpi-forum.org/