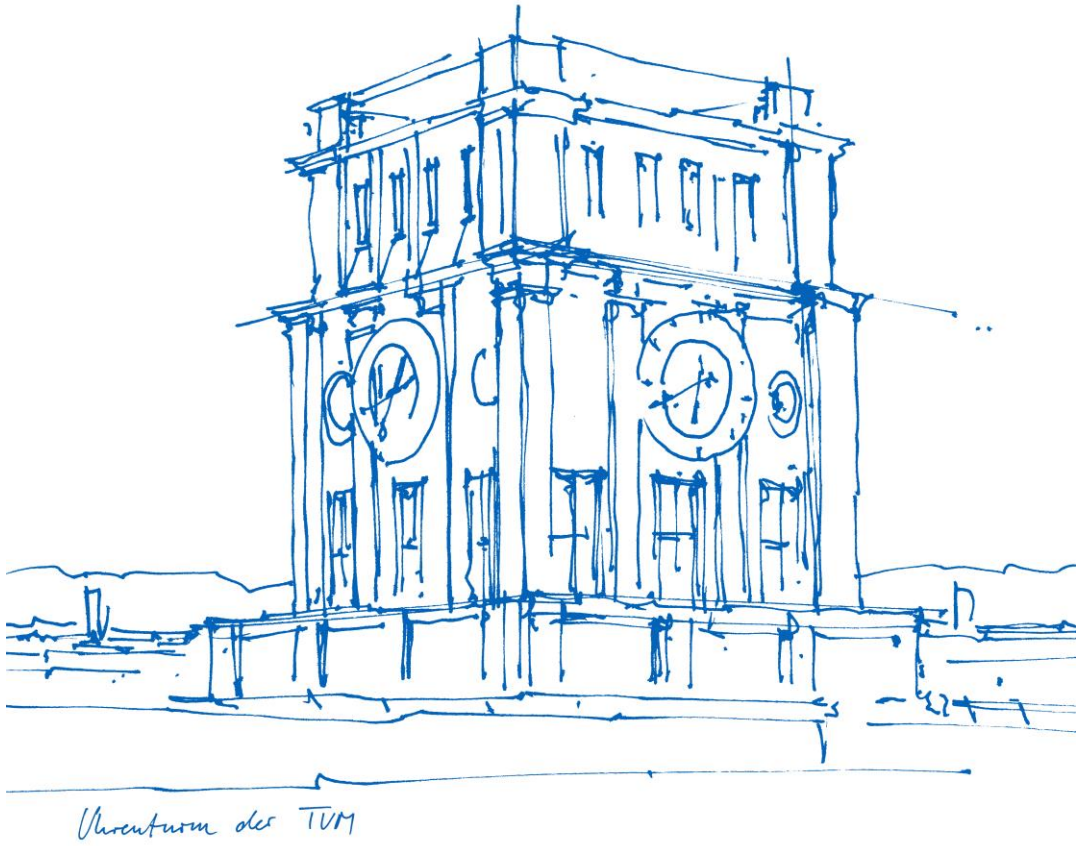
The State of MPI
Current Standard and Future Plans

Martin Schulz
Chair for Computer Architecture and Parallel Systems
Technical University of Munich

NHR PerfLab Seminar Series, June 30th 2026



The Message Passing Interface (MPI)

Designed in 1992, based on previous experiences with message passing libraries

- MPI 1.0 first ratified in 1994
- Most current version 5.0 (for the most backward compatible)
- All documents at: http://www.mpi-forum.org/
- From the 25 year symposium: https://www.mcs.anl.gov/mpi-symposium/

Initial functionality

- Send and receive operations for point to point communication
- Collective operations among groups of processes
- Process group management
- Supports C and Fortran (later up to Fortran 2008)

Independent processes with their own code

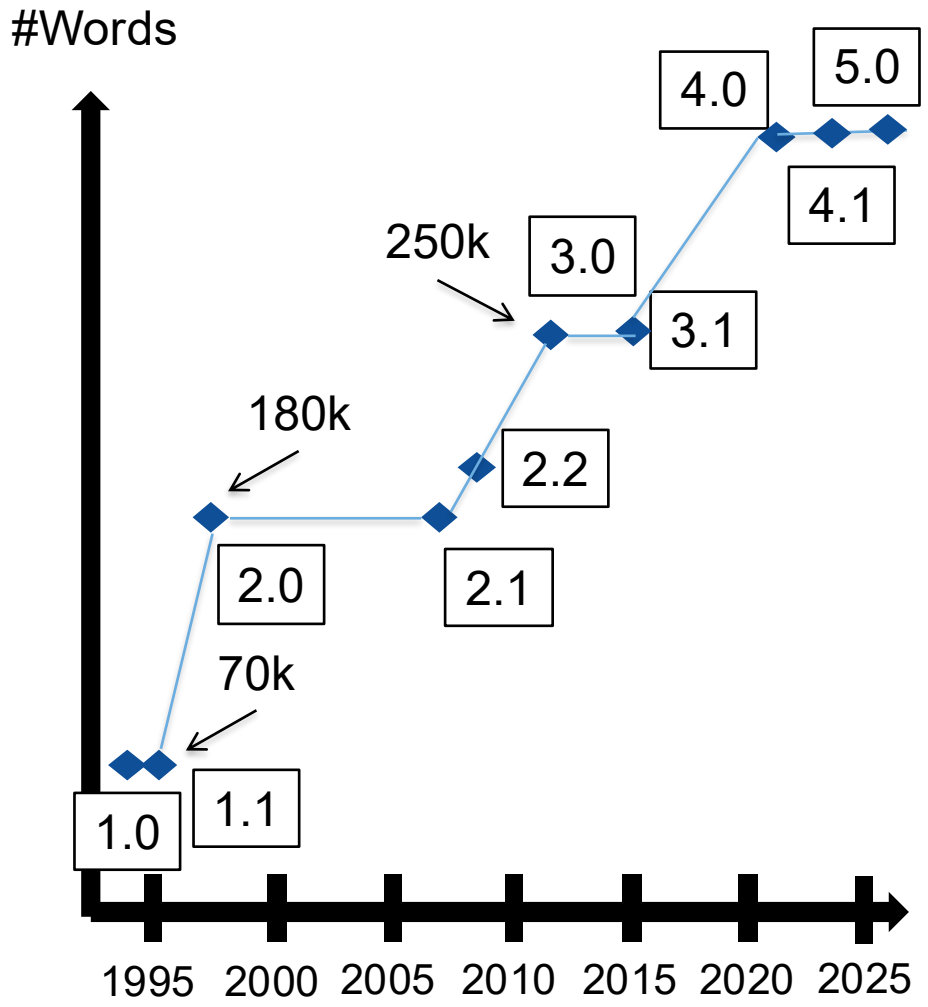
- By default MPI is a library without an ABI (*)
- Compiled against the same/matching MPI implementation
- Different data type representations on each node
- Started independently using a non-specified mechanism

The Growth of MPI

- MPI 1.0 May 1994 – 228 pages
- MPI 1.1 Nov 1995 – 238 pages
- MPI 2.0 Nov 1997 – 608 pages

10 year break

- MPI 2.1 June 2008 – 608 pages
- MPI 2.2 Sep 2009 – 647 pages
- MPI 3.0 Sep 2012 – 852 pages
- MPI 3.1 June 2015 – 868 pages
- MPI 4.0 June 2021 – 1139 pages
- MPI 4.1 Nov. 2023 – 1166 pages
- MPI 5.0 June 2025 – 1189 pages



The MPI Forum Drives MPI
https://www.mpi-forum.org/

Standardization body for MPI

- Discusses additions and new directions
- Oversees the correctness and quality of the standard
- Represents MPI to the community
- Chair (M. Schulz), Secretary (W. Bland), Editor (W. Gropp), Treasurer (B. Smith)

Open membership

- Any organization is welcome to participate
- Consists of working groups and the actual MPI forum
- Voting meetings (typically) 4 times each year (2x virtual, 2x hybrid)
  - 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

Technical work driven by the working groups

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

Typical Way New Features Get Added to MPI

1. New items brought to a matching working group for discussion

2. Creation of preliminary proposal

3. Socializing of idea driven by the WG
   Through community discussions, user feedback, publications, …

   Development of full proposal
   In many cases accompanied with prototype development work

4. MPI forum reading/voting process
   - One reading
   - Two votes

   Slow and consensus driven process

5. Once enough topics are completed:
   Publication of a new standard

How Can You Participate?

1. Follow the MPI Forum website and git presence
   - Some parts are protected, don't be shy to ask for access
2. Follow the MPI Forum email list(s)
   - Easy sign-up on the MPI Forum webpage
3. Provide feedback to the standard:
   - https://www.mpi-forum.org/comments/
4. Join a working group
   - All information on the website
   - Introduce yourself to the WG chair(s)
5. Introduce your own proposal to the WG
   - Start with discussions in the WG
   - Get feedback
   - Can also help shepherd/implement existing proposals
6. Volunteer for one of the WG/officer positions after a while

Join us:
www.mpi-forum.org

Evolution of MPI

Initial functionality in MPI 1.x

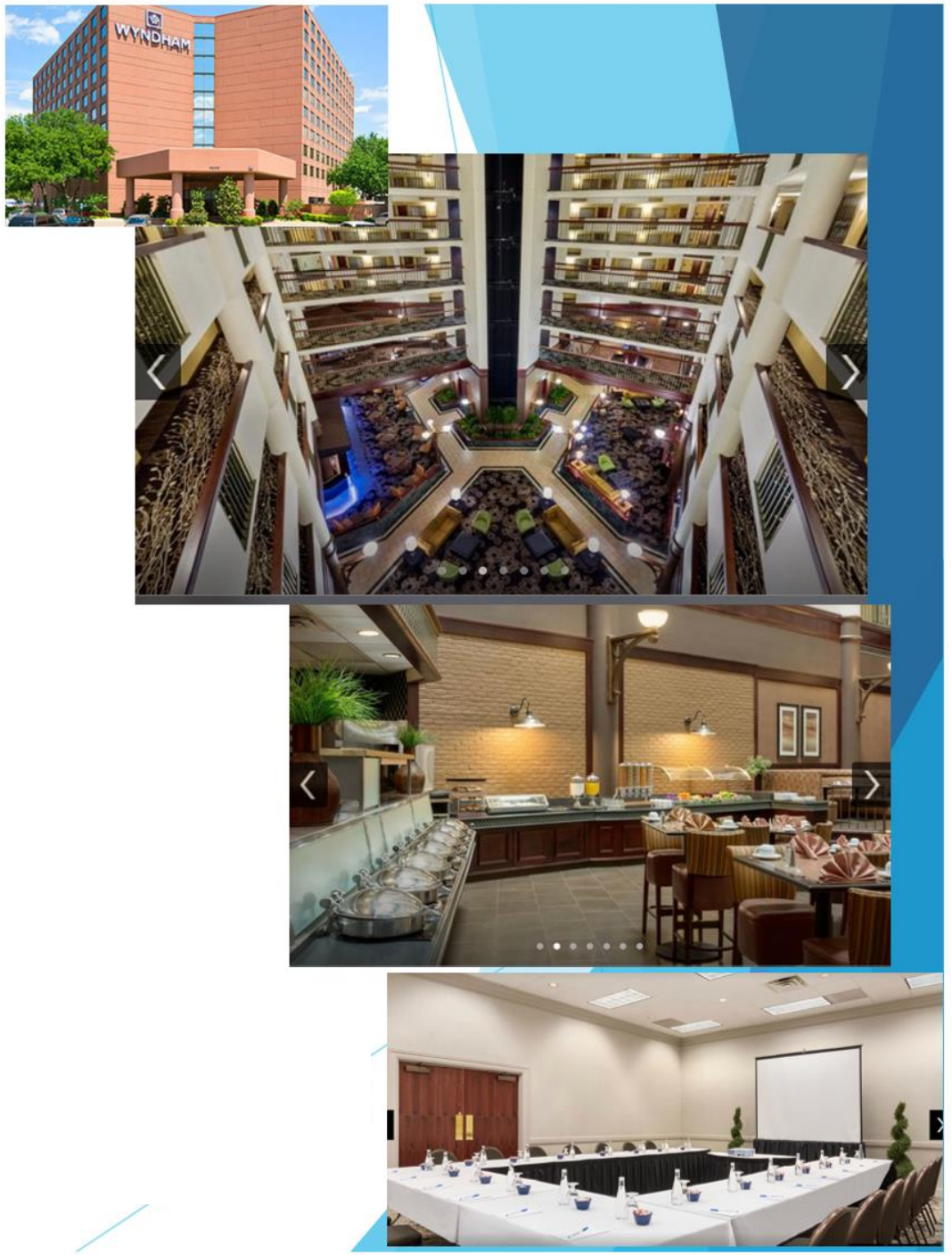
- Send/Recv
  (incl. non-blocking / persistent)
- Blocking collectives
- Group management

Some major additions for MPI 2.x

- Dynamic Processes / Spawns
- Remote memory access (RMA)
- Support for parallel I/O

Some major additions for MPI 3.x

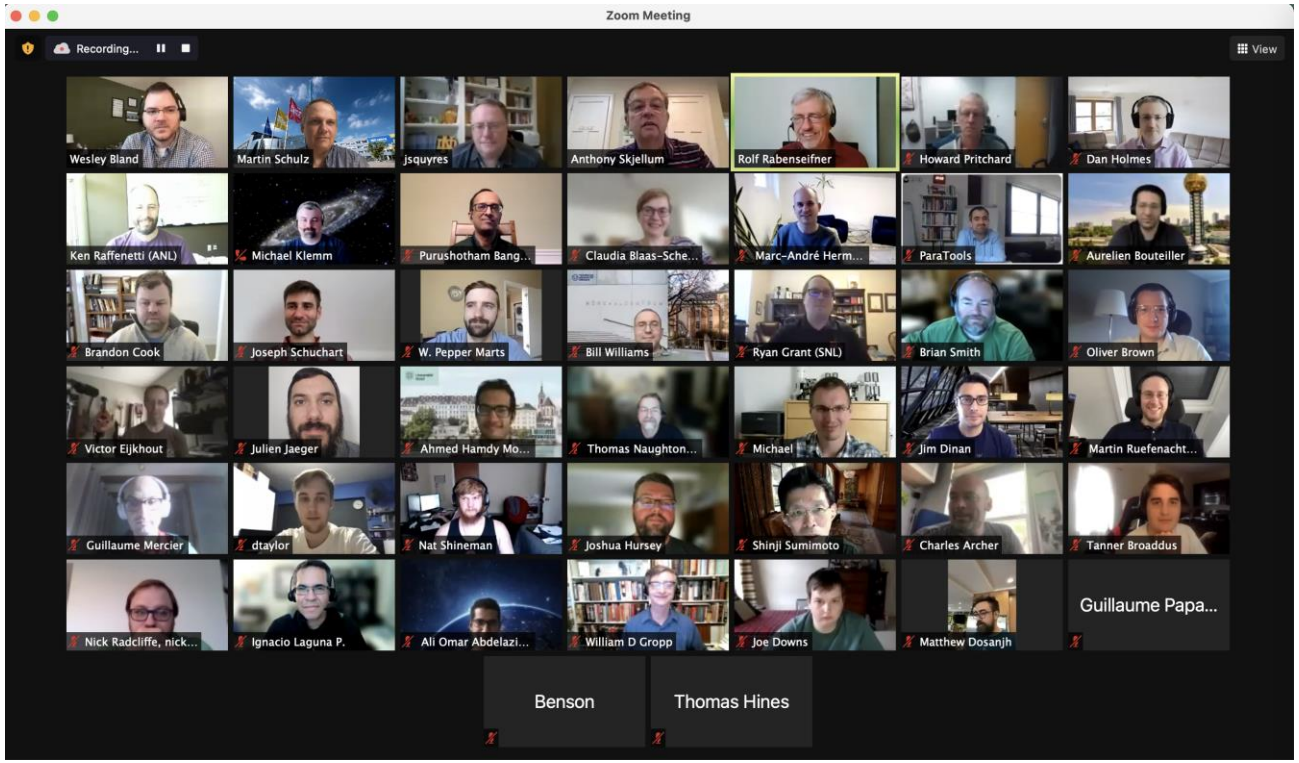
- Nonblocking collectives
- Neighborhood collectives
- New language bindings



Evolution of MPI

MPI 4.0 was published 2.5 years ago (in the middle of Corona)

- Solution for “Big Count” operations
- Partitioned Communication
- Persistent Collectives
- Improved Error Handling
- Topology Solutions
- New init options via MPI Sessions
- And much more …



The “Embiggenment“

Big Count aka. Embiggenment

Problem: in previous 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 included:

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

Last option was selected

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

Example: MPI_Recv

Language Independent Binding

MPI_RECV(buf, count, datatype, source, tag, comm, status)

OUT	buf	initial address of receive buffer (choice)
IN	count	number of elements in receive buffer (non-negative integer)
IN	datatype	datatype of each receive buffer element (handle)
IN	source	rank of source or MPI_ANY_SOURCE (integer)
IN	tag	message tag or MPI_ANY_TAG (integer)
IN	comm	communicator (handle)
OUT	status	status object (status)

MPI 3.1 Version

C binding

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,
             int tag, MPI_Comm comm, MPI_Status *status)

int MPI_Recv_c(void *buf, MPI_Count count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm, MPI_Status *status)
```

BigCount Version

Fortran 2008 binding

```
MPI_Recv(buf, count, datatype, source, tag, comm, status, ierror)
    TYPE(*), DIMENSION(..) :: buf
    INTEGER, INTENT(IN) :: count, source, tag
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Status) :: status
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_Recv(buf, count, datatype, source, tag, comm, status, ierror) !(_c)
    TYPE(*), DIMENSION(..) :: buf
    INTEGER(KIND=MPI_COUNT_KIND), INTENT(IN) :: count
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    INTEGER, INTENT(IN) :: source, tag
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Status) :: status
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran binding

```
MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS(MPI_STATUS_SIZE),
            IERROR
```

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_all
MPI_File_write_at_all_begin
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_Imrecv
MPI_Ineighbor_allgather
MPI_Ineighbor_allgatherv
MPI_Ineighbor_alltoall
MPI_Ineighbor_alltoallv
MPI_Ineighbor_alltoallw
MPI_Ireduce_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_Recv
MPI_Recv_init
MPI_Reduce_local
MPI_Reduce_scatter
MPI_Reduce_scatter_block
MPI_Rget
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 generated via embedded Python

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

```
\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}
```

Consequences

- Slightly different rendering
- More consistency
- Uncovered errors

New opportunities

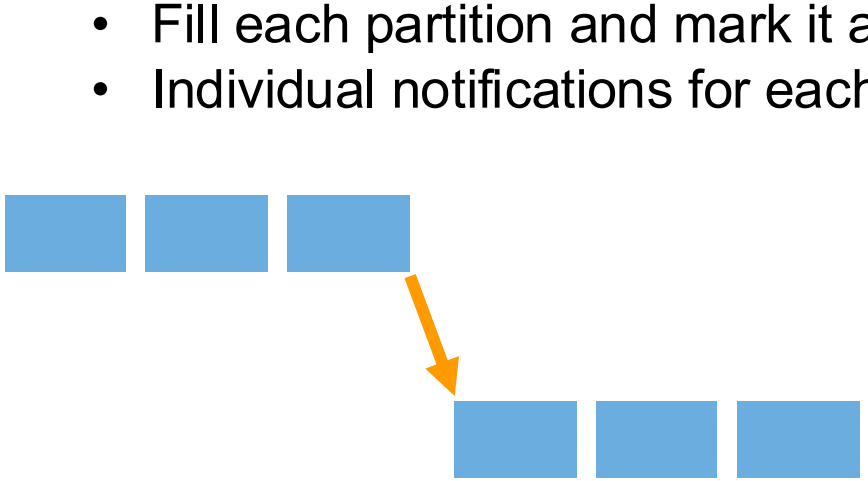
- API / Query mechansism is being developed
- Machine readable description of all MPI routines
- Automatic extraction of the interface
- Eases future standard-wide changes
- Enables better tool support (e.g., generation of PMPI tools)

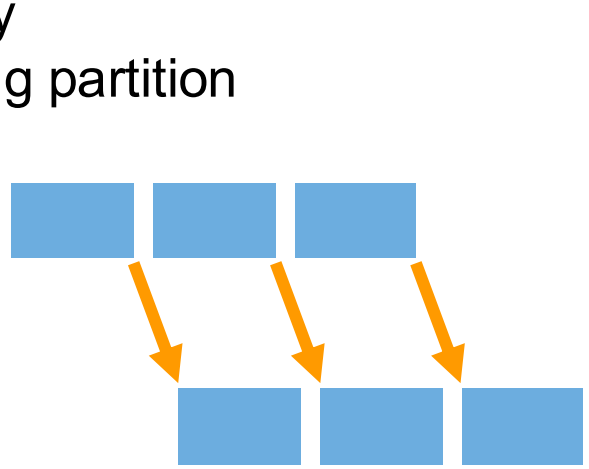
Partitioned Communication

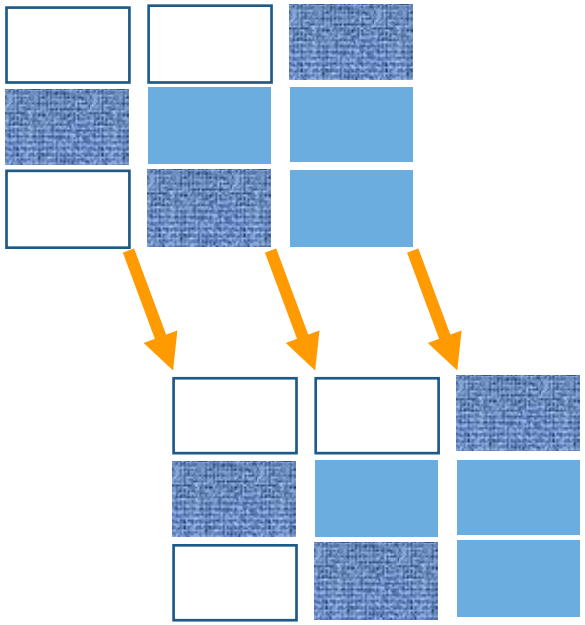
Partitioned Communication

Core idea – efficient highly concurrent communication

- Built on the concept of persistent P2P communication
- Send and receive buffers are split into (possibly different) partitions
  - Fill each partition and mark it as ready
  - Individual notifications for each arriving partition







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

- May be driven from light weight environments, without entire MPI stack
- May need additional synchronization to trigger message transfer safely

Partitioned Communication for Thread Support

```
MPI_Psend_init(..., &request);
for (...) {
  MPI_Start(&request);
  #pragma omp parallel
  {
          kernel(..., request);
  }
  MPI_Wait(&request);
}
MPI_Request_free(&request);
```

```
Thread:

kernel(..., MPI_Request request)
{
  int i = my_partition[my_id];
  /* Compute and fill partition i then mark ready: */
  MPI_Pready(i, request);
}
```

Heavy weight MPI communication outside of parallel region

- Inside only light-weight triggering (easier thread safety)
- Partition for each thread
- Each thread signals when it is ready
- MPI can optimize for latency or bandwidth (or shift)

Persistent Collectives

Persistent Collective Operations

Use-case: a collective operation is done many times in an application

- The specific sends and receives represented never change (size, type, lengths, transfers)
- Opportunities
  - Fixed cost for making optimizations can be amortized
  - Static resource allocation can be done
  - Special limited hardware can be allocated if available

Basics

- Mirror regular nonblocking collective operations
- For each nonblocking MPI collective, MPI now has a persistent variant
- For every MPI_I<coll>, add MPI_<coll>_init
- Parameters are identical to the corresponding nonblocking variant
  - Plus additional MPI_INFO parameter
- All arguments “fixed” for subsequent uses
- Persistent collective operations cannot be matched with other collective calls

Example

```
for (i = 0; i < MAXITER; i++) {
    compute(bufA);
    MPI_Ibcast(bufA, …, rowcomm, &req[0]);
    compute(bufB);
    MPI_Ireduce(bufB, …, colcomm, &req[1]);
    MPI_Waitall(2, req, …);
}
```

Nonblocking collectives API

```
MPI_Bcast_init(bufA, …, rowcomm, &req[0]);
MPI_Reduce_init(bufB, …, colcomm, &req[1]);
for (i = 0; i < MAXITER; i++) {
    compute(bufA);
    MPI_Start(req[0]);
    compute(bufB);
    MPI_Start(req[1]);
    MPI_Waitall(2, req, …);
}
```

Persistent collectives API

Better Error Handling

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.

BUT: Not full fault tolerance for MPI!

Topology Solutions

New Ways to Adapt to Topologies

New systems are very hierarchical

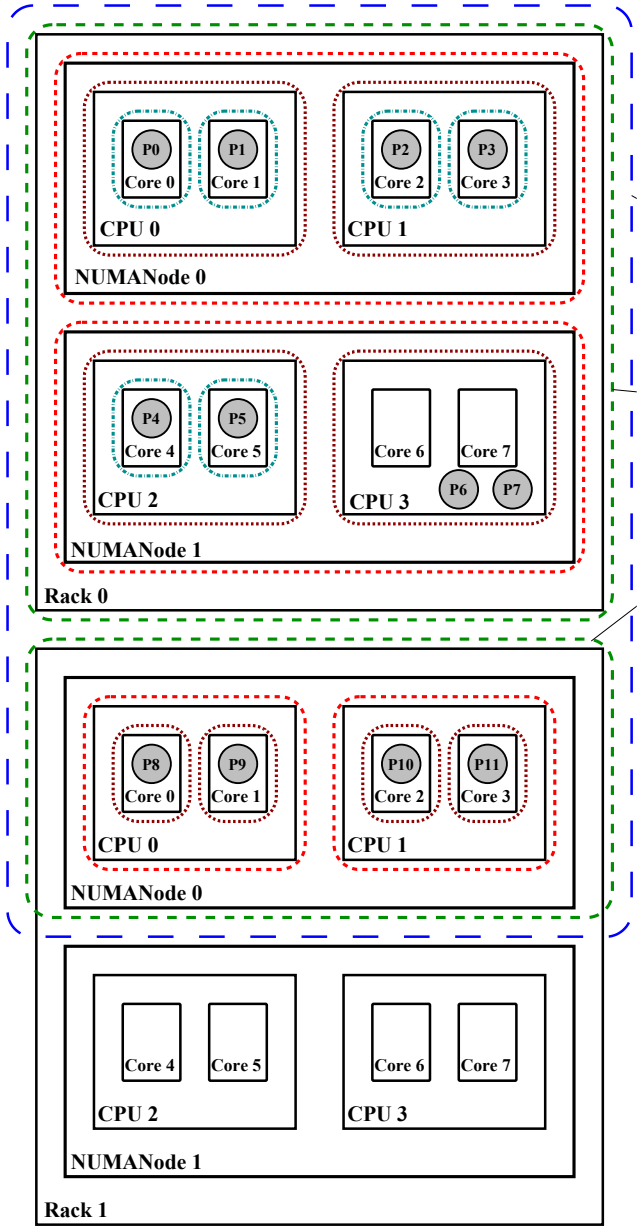
- On node and whole system
- Application mapping is critcal
- Need topology-aware communicators

Guided Mode

- MPI_COMM_SPLIT_TYPE
- Special split type with info key to specify level

Unguided Mode

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

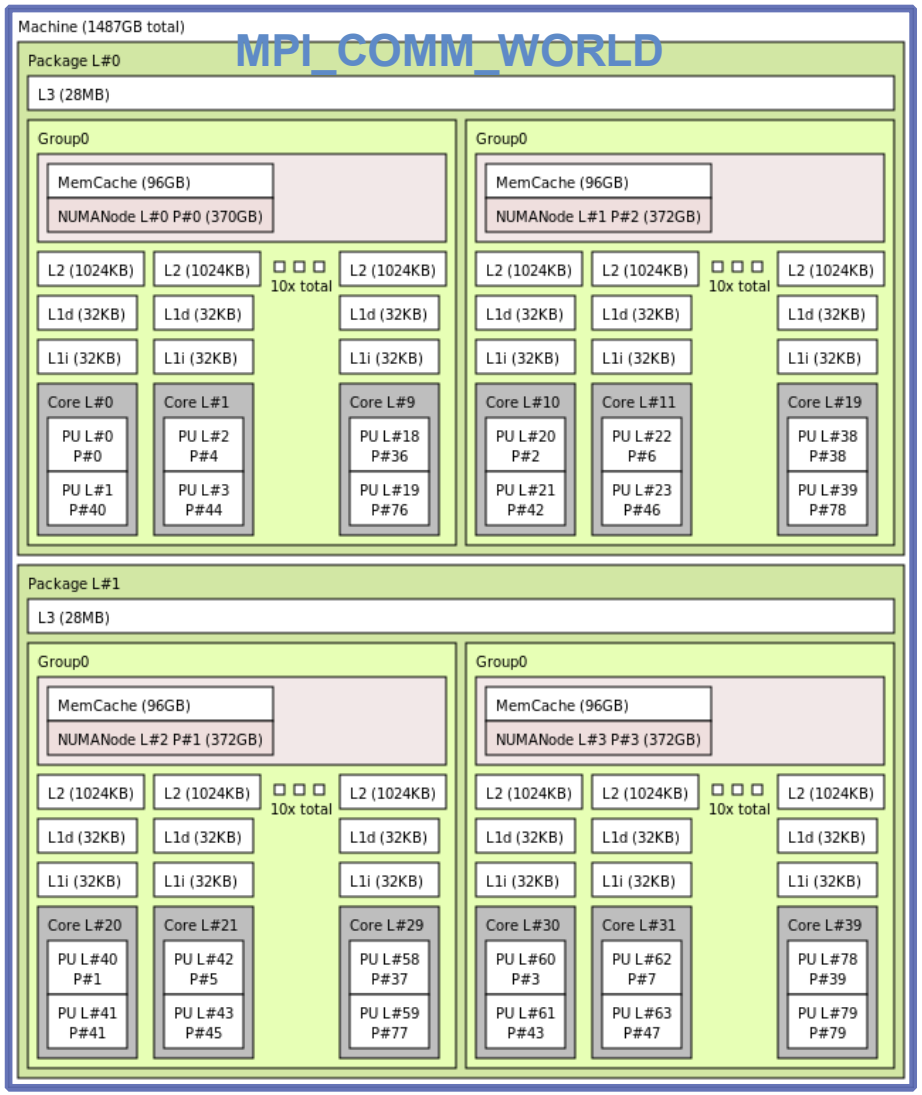


Graphics/Example from Guillaume Mercier, INRIA

Example

MPI_Init(…);

MPI_Comm_rank(MPI_COMM_WOLRD, &rank);



Graphics/Example from Guillaume Mercier, INRIA

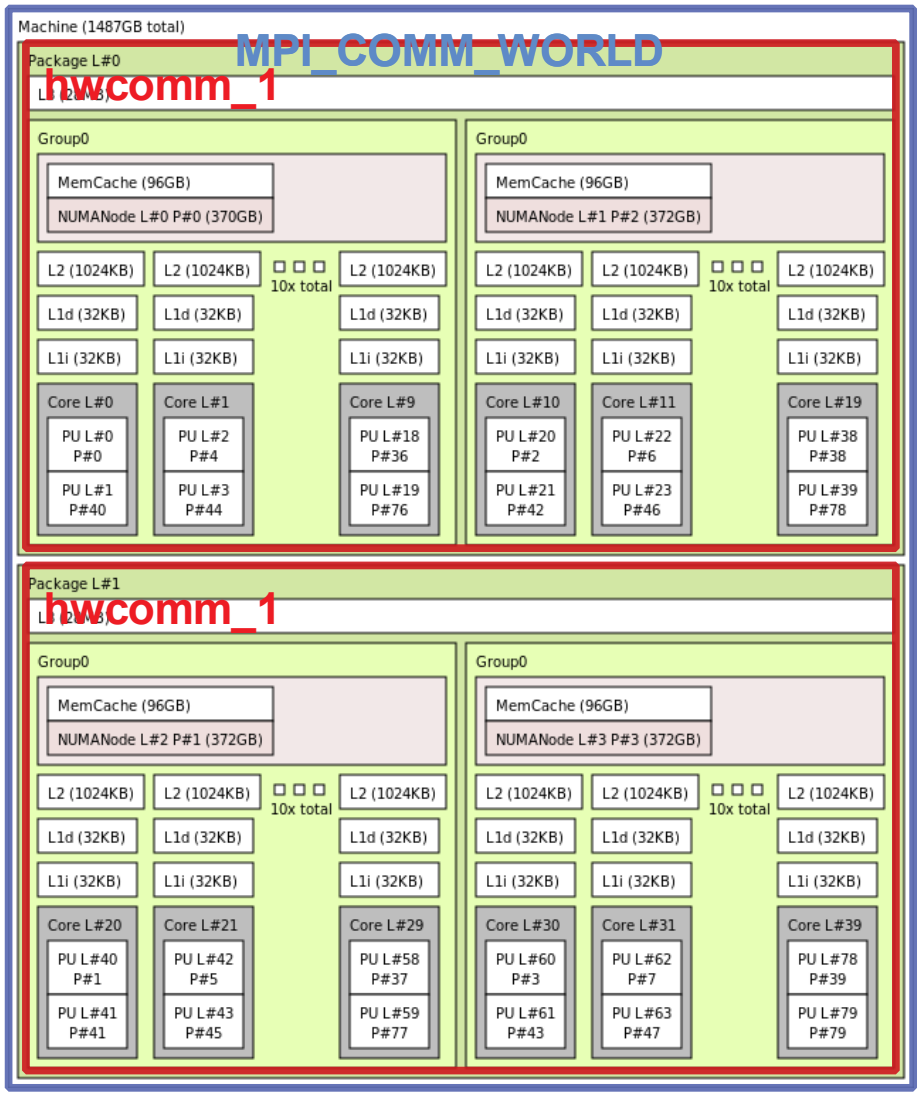
Example

MPI_Init(…);

MPI_Comm_rank(MPI_COMM_WOLRD, &rank);

MPI_Comm_split_type(MPI_COMM_WORLD,

MPI_COMM_TYPE_HW_UNGUIDED,
rank,info,&hwcomm_1);



Graphics/Example from Guillaume Mercier, INRIA

Example

MPI_Init(…);

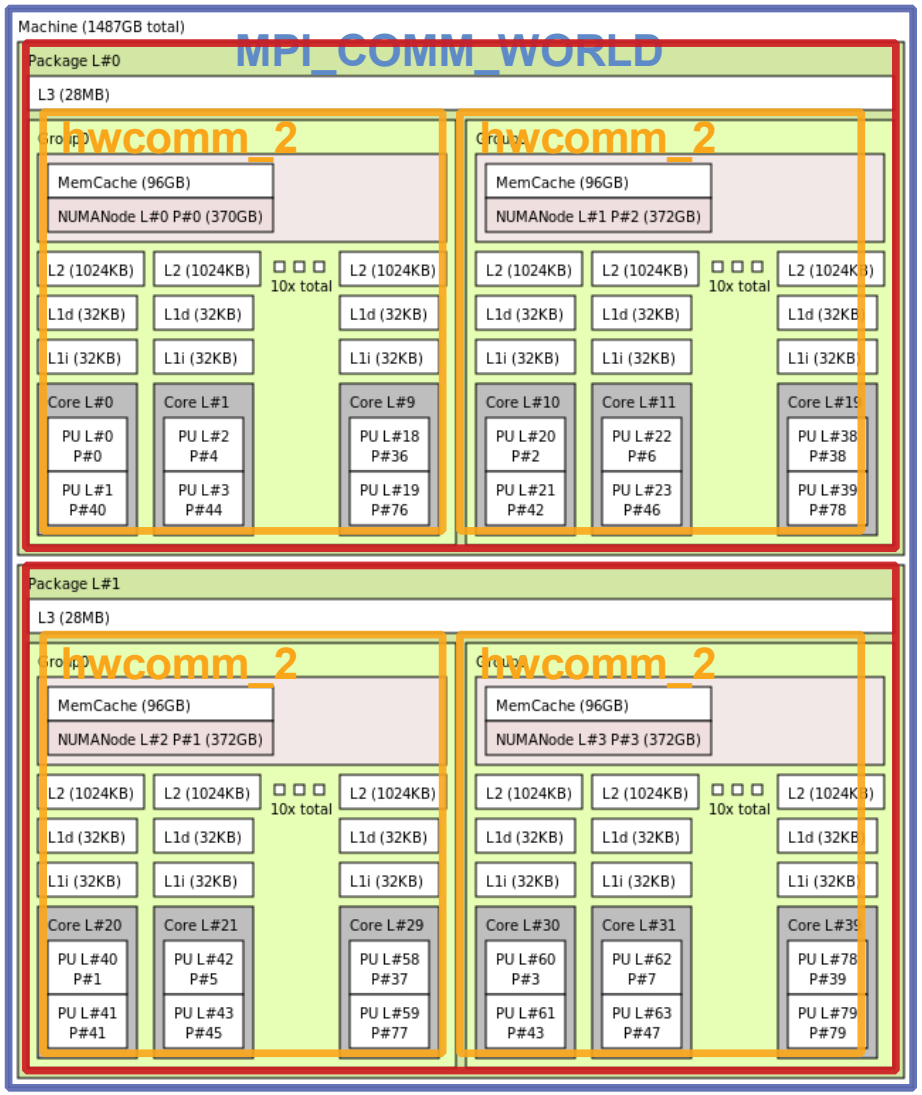
MPI_Comm_rank(MPI_COMM_WOLRD, &rank);

MPI_Comm_split_type(MPI_COMM_WORLD,

MPI_COMM_TYPE_HW_UNGUIDED,
rank,info,&hwcomm_1);

MPI_Comm_split_type(hwcomm_1,

MPI_COMM_TYPE_HW_UNGUIDED,
rank,info,&hwcomm_2);



Graphics/Example from Guillaume Mercier, INRIA

MPI Sessions

MPI Sessions

Attacking some fundamental problems in MPI

- MPI_COMM_WORLD is a very static resource
  - Minimized the complexity
- MPI_COMM_WORLD is immutable
  - Avoids many issues and reduces overhead (and made it not PVM ☺)
- MPI has no ability to isolate resources
  - This was not necessary in the past, very little to isolate
- MPI has no ability to “talk” to the runtime system
  - Explicit choice to improve portability, separation of concerns and matched HPC setups
- MPI is seen as not supporting new communities and industrial applications
  - HPC was the initial target and is still the main area

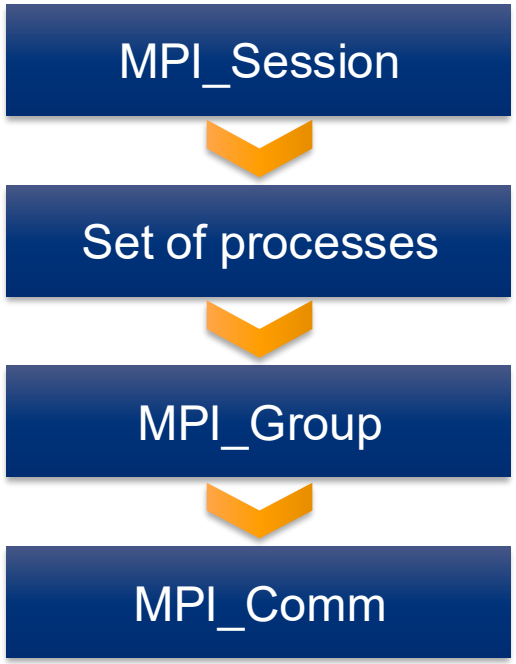
Question: How can we overcome this, without compromising MPI?

- Backwards compatible, but that is only part of it
- Same look and feel of MPI, maintain the learning curve
- Enabe code reuse for existing codes

MPI Sessions

Instead of MPI_Init / MPI_COMM_WORLD:

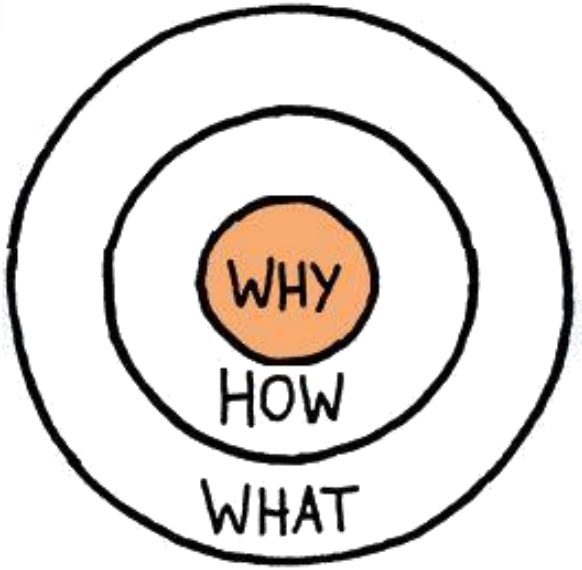
1. **Get local access to the MPI library**
   Get a Session Handle
2. **Query the underlying run-time system**
   Get a "set" of processes
3. **Determine the processes you want**
   Create an MPI_Group
4. **Create a communicator with just those processes**
   Create an MPI_Comm



What does this do?

- Deliver runtime information of (changing) information to the MPI library
- Enables the ability to provide resource isolation between sessions
- Eliminate the need for a static resource MPI_COMM_WORLD

It's a starting point!



Evolution of MPI

MPI 4.0 was published 2.5 years ago (in the middle of Corona)

- Solution for “Big Count” operations
- Partitioned Communication
- Persistent Collectives
- Improved Error Handling
- Topology Solutions
- New init options via MPI Sessions
- And much more …

MPI 4.1 was published November 2023

- Clean-Up & terminology adjustments
- Automatic buffer for MPI_Bsend
- Support for different memory kinds, including GPU memory

What's New in MPI 4.1? Updates

MPI 4.1 is a minor update to MPI, mostly bug fixes and clarifications

Many Clarifications

- Terms and behavior of routines in some corner cases.
- See the change log for specifics.

Errata items

- Revert change to MPI_CART_COORDS made by mistake in MPI 4.0
- MPI_STATUS_SET_ELEMENTS for large count added

Deprecated (MPI features that might be removed in a later version)

- Some obscure functions deprecated (MPI_TYPE_SIZE_X and such)
- MPI_HOST (attribute key) deprecated
- mpif.h deprecated (Fortran programmers should use the mpi or mpi_f08 module)

What's New in MPI 4.1? New Features

Added functions to query/set status fields

- E.g., MPI_STATUS_GET_SOURCE. Can still use direct access.

Improved Bsend support

- Automatic, "unlimited" buffering for BSEND added
- New buffer attach/detach for communicators and sessions.
- New routines to complete communication (e.g., MPI_COMM_FLUSH_BUFFER)

Added functions to query status on a request

- E.g., MPI_REQUEST_GET_STATUS_ANY
- Without freeing the request

Additions to HW Topologies

- Added MPI_COMM_TYPE_RESOURCE_GUIDED for MPI_COMM_SPLIT_TYPE
- Added MPI_GET_HW_RESOURCE_INFO

Added routines to remove error codes, classes, and strings

- For those added by the user

New in MPI 4.1: Allocator Kind Info

Use MPI info to provide users with a portable solution to:

1. Detect whether accelerator memory is supported by the MPI library
2. Request support for accelerator memory from the MPI library (when using Sessions)
3. Constrain usage of accelerator memory to specific communicators, windows, etc.

mpi_request_memory_alloc_kind

- Request support for memory allocator kind from the MPI library

mpi_assert_memory_alloc_kind

- Assert memory kinds used by the application on the given MPI object

mpi_memory_alloc_kind

- Memory kinds supported by the MPI library

Slide by Jim Dinan, NVIDIA

New in MPI 4.1: Allocator Kind Info

Use MPI info to provide users with a portable solution to:

1. Detect whether accelerator memory is supported by the MPI library
2. Request support for accelerator memory from the MPI library (when using **Sessions**)
3. Constrain usage of accelerator memory to specific communicators, windows, etc.

mpi_request_memory_alloc_kind

- Request support for memory allocator kind from the MPI library

mpi_assert_memory_alloc_kind

- Assert memory kinds used by the application on the given MPI object

mpi_memory_alloc_kind

- Memory kinds supported by the MPI library

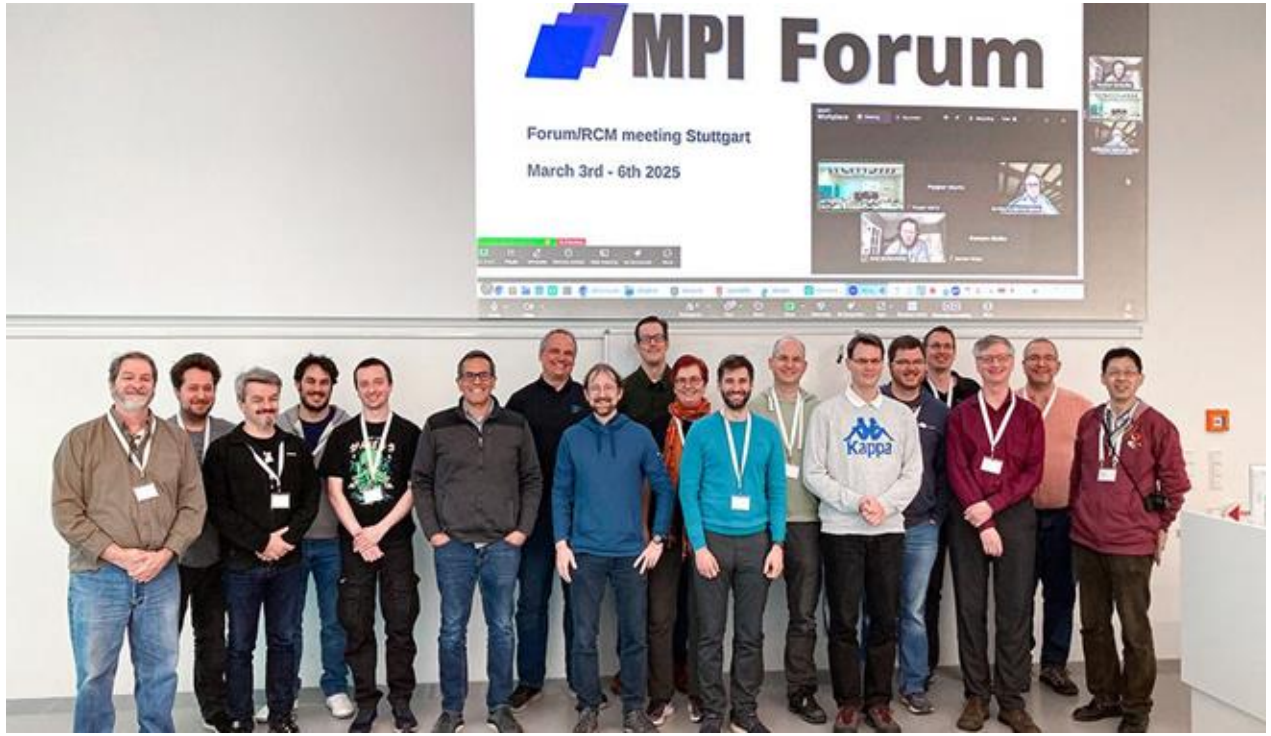
Slide by Jim Dinan, NVIDIA

Request Support for CUDA Allocated Memory

```
bool cuda_aware = false;
int len = MPI_MAX_INFO_VAL, flag = 0;
char *val = malloc(MPI_MAX_INFO_VAL);
MPI_Info info;

MPI_Info_create(&info);
MPI_Info_set(info, “mpi_memory_alloc_kinds”, “cuda:device”);
MPI_Session_init(info, MPI_ERRORS_ARE_FATAL, &session);
MPI_Info_free(&info);

MPI_Session_get_info(session, &info);
MPI_Info_get_string(info, “mpi_memory_alloc_kinds”, &len, val, &flag);

// Check mpi_memory_alloc_kind for “cuda:device”
while (flag && (kind = strsep(&val, ",")) != NULL) {
  if (strcasecmp(kind, "cuda:device") == 0) {
    cuda_aware = true;
    break;
  }
}
```

Slides by Jim Dinan, NVIDIA

Evolution of MPI



MPI 4.0 was published 2.5 years ago (in the middle of Corona)

- Solution for “Big Count” operations
- Partitioned Communication
- Persistent Collectives
- Improved Error Handling
- Topology Solutions
- New init options via MPI Sessions
- And much more …

MPI 4.1 was published November 2023

- Clean-Up & terminology adjustments
- Automatic buffer for MPI_Bsend
- Support for different memory kinds, including GPU memory

MPI 5.0 was published June 2025 (previously targeted as MPI 4.2)

- Introduction of an ABI

API vs. ABI

MPI in, its conception, is an API Standard

- Defines the source code behavior in C (C++) and Fortran
- The compiled representation of MPI features is implementation-defined

API

```
int MPI_Bcast(void * buffer, int count, MPI_Datatype d, int root, MPI_Comm c);
MPI_Datatype and MPI_Comm are unspecified types
```

ABI

```
typedef struct ompi_datatype_t * MPI_Datatype; // Open MPI family
typedef int MPI_Datatype; // MPICH family
```

Different ABIs for different MPI implementations

- There are two major families: MPICH vs. Open MPI
- If you compile with one MPI implementation, you MUST run with the same

Consequences

- Difficult software management
- Recompilation of libraries and tools

Based on slides from J. Hammond, NVIDIA

The MPI ABI

Defined starting MPI 5.0

- Defined way to specify all types, constants, …
- Any application compiled against an ABI can run with any MPI using the ABI

Why?

- Third-party language support, e.g. Python, Julia, Rust, etc.
- Package distribution, e.g. Spack, Apt, etc.
- Tools become implementation-agnostic
- Containers
- More efficient testing (build only once)

Why not earlier?

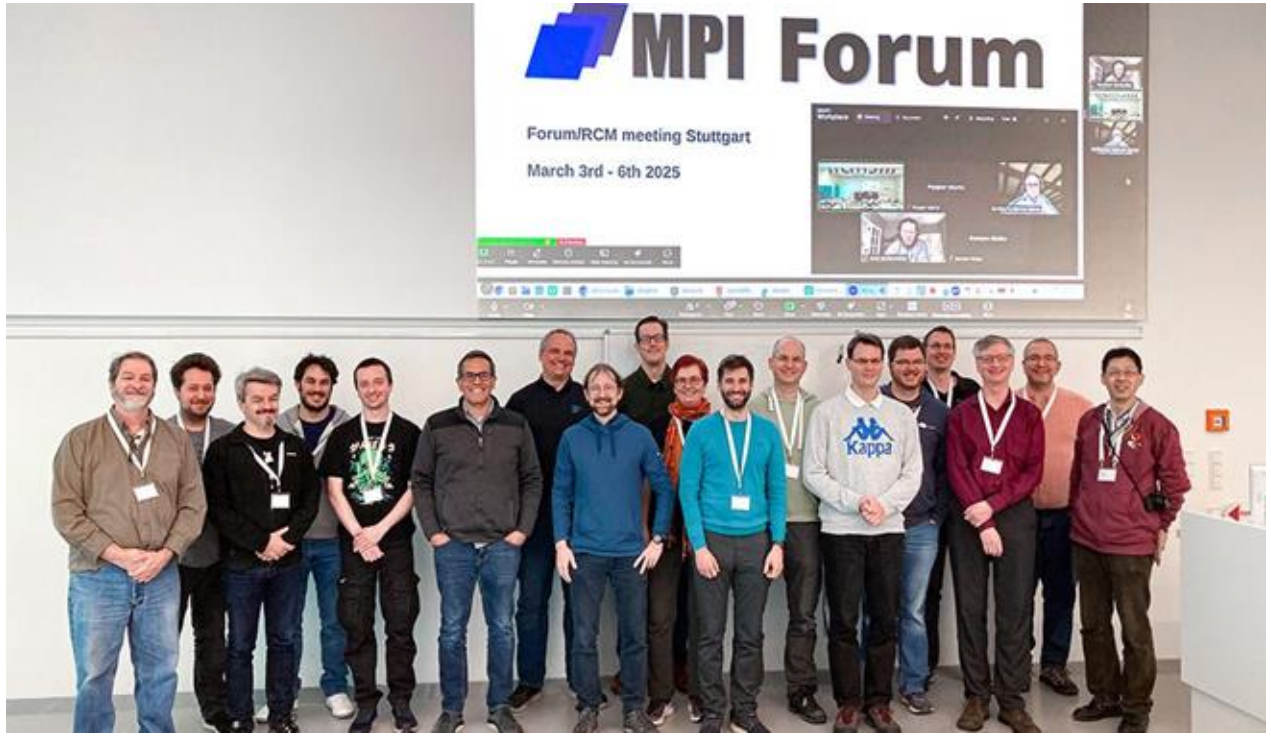
- Architectural reasons not to are gone
- Two platform ABIs cover >90% of HPC platforms

MPI ABI Packaging / Usage

- New header: `abi/mpi.h`
- Special library name to link against: `{lib}mpi_abi.ext`
- The ABI is versioned independently from the API (starting with v1.0)

Based on slides from J. Hammond, NVIDIA

Evolution of MPI



MPI 4.0 was published 2.5 years ago (in the middle of Corona)

- Solution for “Big Count” operations
- Partitioned Communication
- Persistent Collectives
- Improved Error Handling
- Topology Solutions
- New init options via MPI Sessions
- And much more …

MPI 4.1 was published November 2023

- Clean-Up & terminology adjustments
- Automatic buffer for MPI_Bsend
- Support for different memory kinds, including GPU memory

MPI 5.0 was published June 2025 (previously targeted as MPI 4.2)

- Introduction of an ABI

Moving Towards MPI 6.0

Building on the new concepts of MPI 4.0/4.1, extend the ABI as needed

- Partitioned Communication, MPI Sessions, Memory kinds
- Increased presence of persistency, introduction of streams

New features that are under discussion

- Better support for accelerated architectures

Partitioned Communication for Thread Support

```
MPI_Psend_init(..., &request);
for (...) {
  MPI_Start(&request);
  #pragma omp parallel
  {
          kernel(..., request);
  }
  MPI_Wait(&request);
}
MPI_Request_free(&request);
```

```
Thread:

kernel(..., MPI_Request request)
{
  int i = my_partition[my_id];
  /* Compute and fill partition i then mark ready: */
  MPI_Pready(i, request);
}
```

Extensions to basic concept

- Guarantees for readiness at receiver
- Different send type options
- Collective versions
- Do non persistent version make sense?

Partitioned Communication for Thread Support

```
MPI_Psend_init(..., &request);
for (...) {
  MPI_Start(&request);
  #pragma omp parallel
  {
          kernel(..., request);
  }
  MPI_Wait(&request);
}
MPI_Request_free(&request);
```

```
Thread:

kernel(..., MPI_Request request)
{
  int i = my_partition[my_id];
  /* Compute and fill partition i then mark ready: */
  MPI_Pready(i, request);
}
```

Extensions to basic concept

- Guarantees for readiness at receiver
- Different send type options
- Collective versions
- Do non persistent version make sense?

Extensions to accelerators

- Heavy weight MPI on host process
- Light-weight triggers on accelerators (without OS)
- Requires translation of requests

Requires GPU bindings for MPI

Partitioned Communication for Thread Support

```
MPI_Psend_init(..., &request);
for (...) {
  MPI_Start(&request);
  #pragma omp target
  {
          GPU_kernel(..., request);
  }
  MPI_Wait(&request);
}
MPI_Request_free(&request);
```

```
Accelerator:

kernel(..., MPI_GPU_Request request)
{
  int i = my_partition[my_id];
  /* Compute and fill partition i then mark ready: */
  MPI_Pready(i, request);
}
```

Extensions to basic concept

- Guarantees for readiness at receiver
- Different send type options
- Collective versions
- Do non persistent version make sense?

Extensions to accelerators

- Heavy weight MPI on host process
- Light-weight triggers on accelerators (without OS)
- Requires translation of requests

Requires GPU bindings for MPI

Accelerator Bindings for MPI Partitioned APIs

CUDA and SYCL Language Bindings Under Exploration

- Consequence of moving partitioned triggers to GPU
- Opens many cans of worms (What is a process? Which bindings? …)

```
int MPI_Psend_init(const void *buf, int partitions, MPI_Count count,
              MPI_Datatype datatype, int dest, int tag, MPI_Comm comm,
              MPI_Info info,MPI_Request *request)

int MPI_Precv_init(void *buf, int partitions, MPI_Count count,
              MPI_Datatype datatype, int source, int tag, MPI_Comm comm,
              MPI_Info info, MPI_Request *request)

int MPI_[start,wait][_all](...)
```

Keep host only

Add device bindings

```
__device__ int MPI_Pready(int partition, MPI_Request request)

__device__ int MPI_Pready_range(int partition_low, int partition_high, MPI_Request request)

__device__ int MPI_Pready_list(int length, const int array_of_partitions[], MPI_Request
request)

__device__ int MPI_Parrived(MPI_Request request, int partition, int *flag)
```

Separation of Bindings from MPI Concepts

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

Current efforts

- New C++ interface
- Following OO design principles
- Draft almost complete
- Likely first as side document

Moving Towards MPI 6.0

Building on the new concepts of MPI 4.0/4.1

- Partitioned Communication, MPI Sessions, Memory kinds
- Increased presence of persistency, introduction of streams

New features that are (or IMHO should be) under discussion

- Better support for accelerated architectures
- New language support to enable new communities
- Better integration with task-based runtimes

Proposal for Thread Continuations

Idea: **Treat the completion of an MPI operation as continuation of some activity**
Ability to couple with OpenMP events and dependencies

```
3
void release_event(MPI_Status status, void *data)
{
  omp_event_handle_t event = (omp_event_handle_t)(uintptr_t)data;
  omp_fulfill_event(event);
}
```

```
MPI_Request cont_req;
1 MPIX_Continue_init(&cont_req);

omp_event_handle_t event;
int value;
#pragma omp task depend(out:value) detach(event)
{
  MPI_Request req;
2 MPI_Irecv(&value, ..., &req);
  MPIX_Continue(&req, &release_event, event, MPI_STATUS_NULL, cont_req);
}

#pragma omp task depend(in: value)
4 {
  // process value
}
```

"Callback-based completion notification using MPI Continuations,"
Joseph Schuchart, Christoph Niethammer, José Gracia, George Bosilca, Parallel Computing, 2021.

"MPI Detach - Asynchronous Local Completion,"
Joachim Protze, Marc-André Hermanns, Ali Demiralp, Matthias S. Müller, Torsten Kuhlen. EuroMPI '20.

Improving the Profiling Interface: QMPI

MPI supports the tools via the MPI profiling interface

- Ability to intercept any MPI call, including parameters
- Redirection of MPI calls for new functionality
- Used by many tools

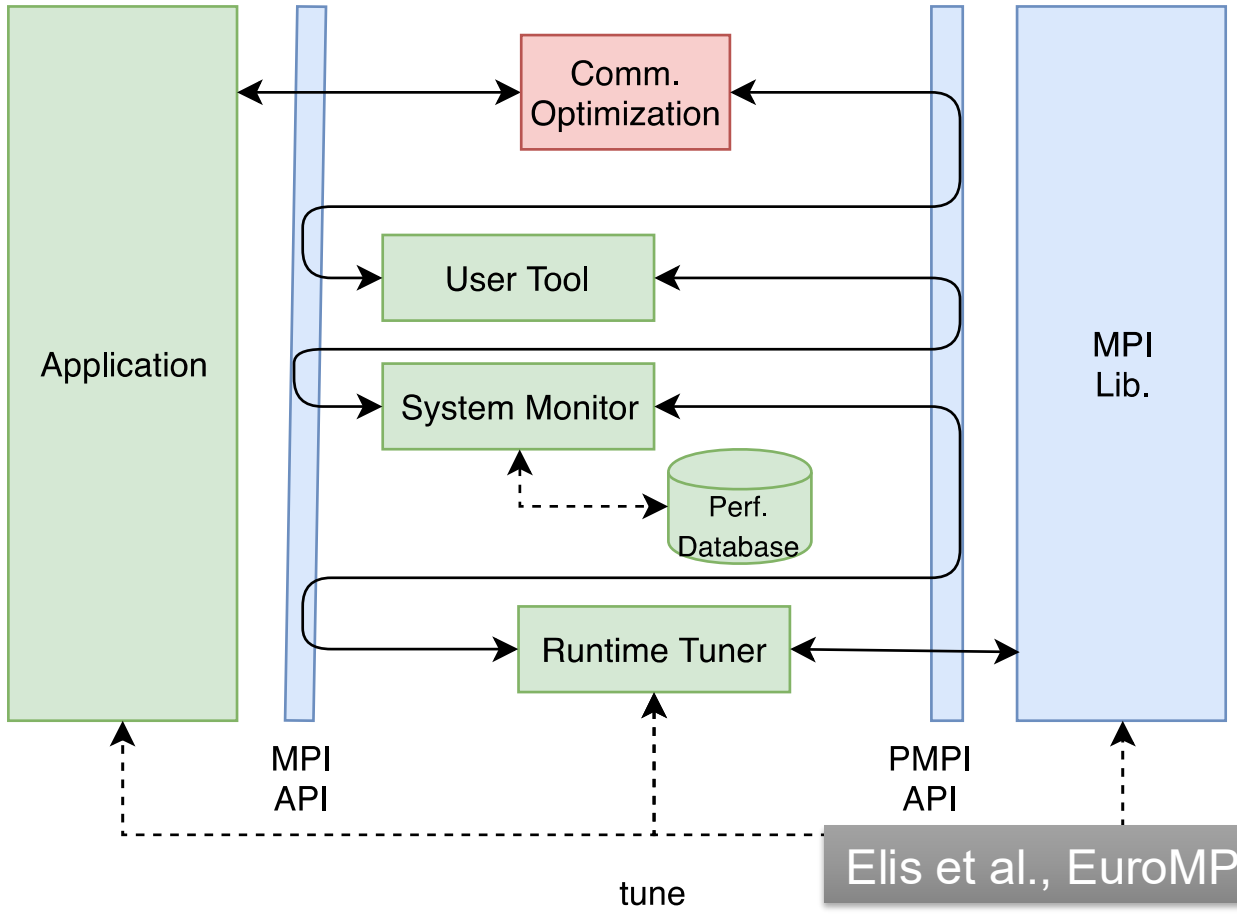
Limitations

- Single tool and user drive
- Usage in Fortran/non-C problematic

Proposal to revamp the interface

- Tools as dynamic wrappers
- Language independent
- Chains of multiple tools

Challenge: Sessions (!)



Elis et al., EuroMPI19

Moving Towards MPI 6.0

Building on the new concepts of MPI 4.0/4.1

- Partitioned Communication, MPI Sessions, Memory kinds
- Increased presence of persistency, introduction of streams

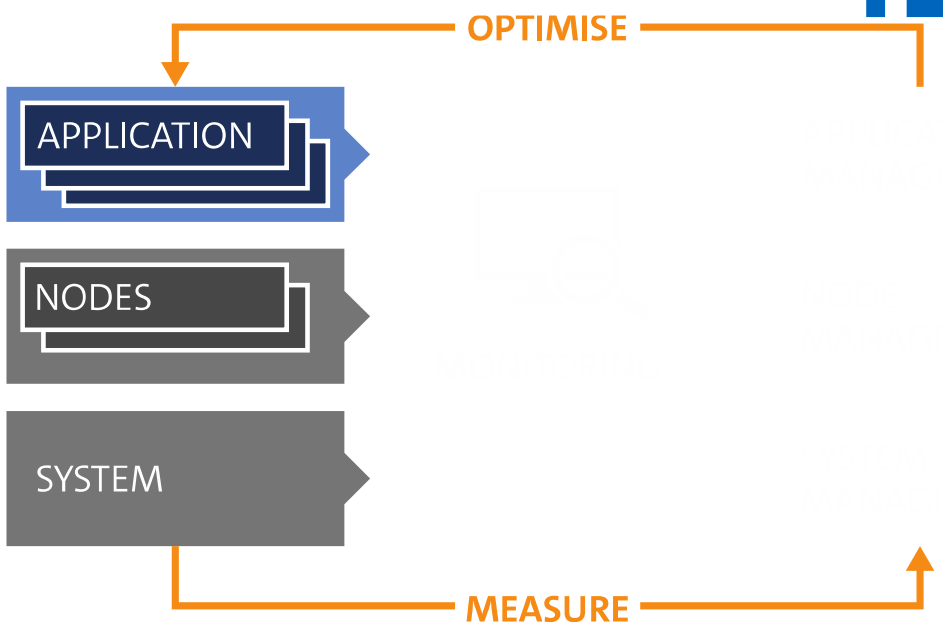
New features that are (or IMHO should be) under discussion

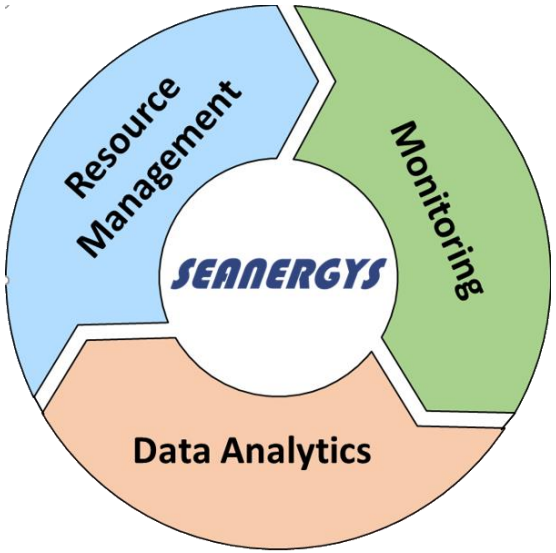
- Better support for accelerated architectures
- New language support to enable new communities
- Better integration with task-based runtimes
- Improved tool support (handles and interceptions)
- Malleability

Malleability / Dynamic Execution

Support for more flexible HPC environments

- Applications to adjust to their sweet-spot
- Systems to adjust based on resources
- Accelerated systems with complex nodes
- Communication with disaggregated accelerators
- Complex workflows





Example: SEANERGYS

- Adaptive systems to increase energy efficiency
- Observe → Predict → Adjust
- Dynamic resources are a key element
  - On-node to steer co-scheduling
  - Cross-node to distribute load and power

EU Grant # 101177590
BMBF #16HPC111
SEANERGYS

EU Grant #955606
BMBF #16HPC014
DEEP-SEA

SPONSORED BY THE
Federal Ministry of Education and Research

EU Grant #956560
BMBF #16HPC039K
REGALE

Time-X #955701
BMBF #16HPC050
TIME-X

Malleability / Dynamic Execution

New user communities

- Going beyond typical application-driven messaging
- Communication in system software
- Commercial applications beyond HPC (“faster sockets”)

Adaptivity is needed to support efficient resource utilization
Adaptivity is needed to manage multiple accelerators
Adaptivity is needed to implement co-scheduling
Adaptivity is needed to arrange complex workflows/workloads

Must be implemented across the entire stack:

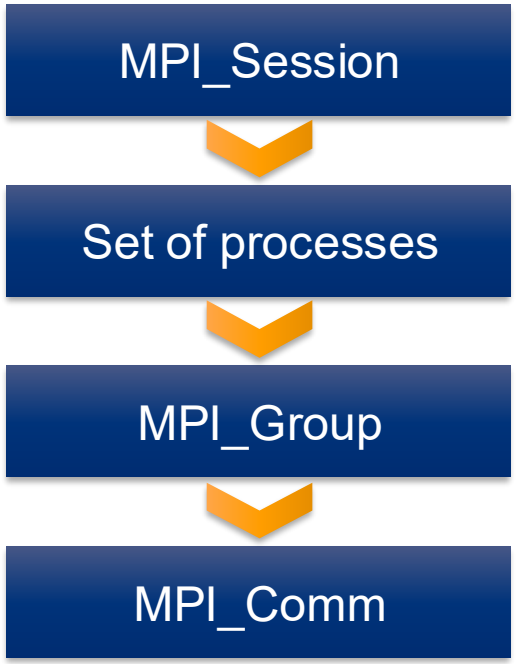
Hardware → firmware → middleware → application

How to change MPI to support malleability?

MPI Sessions

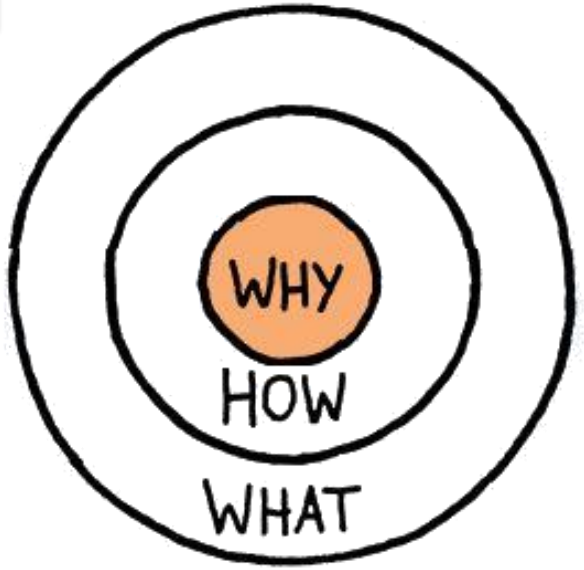
Instead of MPI_Init / MPI_COMM_WORLD:

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



What does this do?

- Deliver runtime information of (changing) information to the MPI library
- Enable resource isolation between sessions
- Eliminate the static resource MPI_COMM_WORLD



Opens the door to malleability in MPI

Building on top of MPI Sessions

Option 1: MPI Session form ”MPI Bubbles”

- “All resources that are derived from a set of resources across a set of MPI processes”
- Implicitly derived from MPI application using sessions
- Within an MPI bubble, normal MPI
- Can invalidate and recreate new bubbles
- New sessions can have new process sets
- Need some kind of versioning

➢ **Medium granularity**

Option 2: Process sets can change

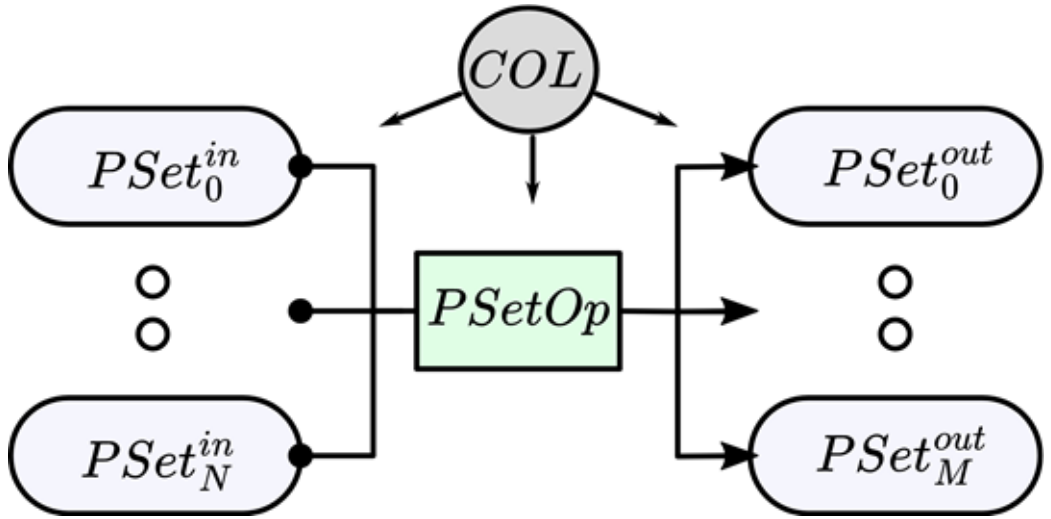
- Names are local to MPI Sessions
- Enable process sets to grow or shrink
- Ability for the runtime to “tell” something to the application
- New routines to manipulate process sets
- Agreement protocol/versioning to agree on new set

➢ **Fine granularity**

Dynamic Processes with Process Sets (DPP)

Operations can modify process sets

- Process set manipulation
- Process set registration after agreement
- Options to transition



Separation of concerns:

- Dynamic Process Management (DPM)
- Dynamic Resource Allocation (DRA)

1. DPM based on PSets and PSetOps

2. DRA based on Collaborative Optimization Language (COL)

Full paper

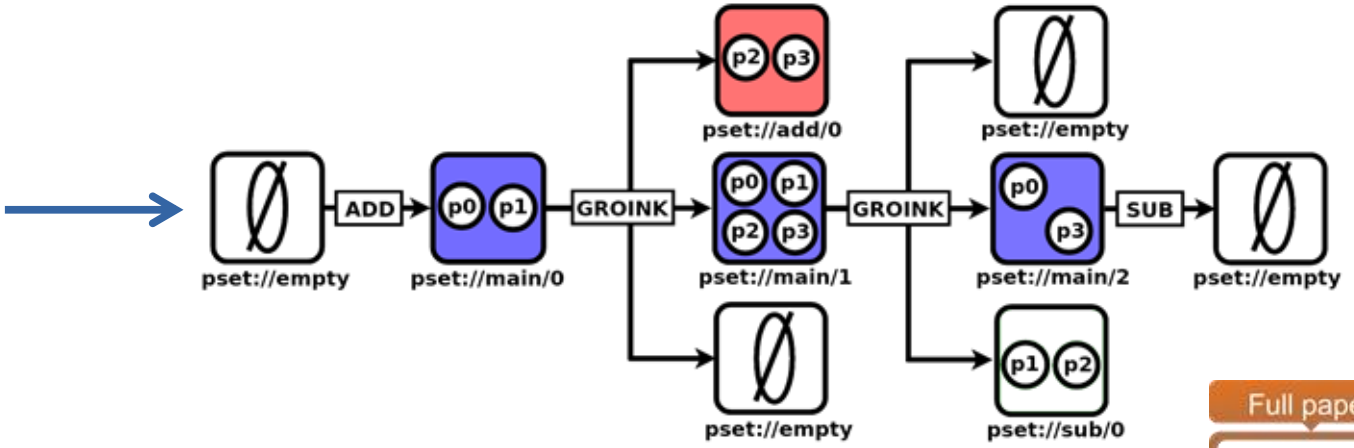
Design Principles of Dynamic Resource Management for High-Performance Parallel Programming Models →

Application Support for DPPs

Example: Dynamic Processes with Process Sets (DPP)

PetSc	LibPFASST	MPDATA3D
Parallel numerical software library	Parallel-in-time integration library	3D semi-Lagrangian multiscale fluid scheme
TS context to solve a nonlinear, time-dependent PDE	2D heat equation with Hypre space parallelisation	Used in EULAG (Eulerian/semi-Lagrangian solver)
P4est	**XBraid**	**DMR-API**
Adaptive Mesh Refinement library	Multigrid parallel-in-time integration	Dynamic resources API for computational kernels
Solve SWE with an augmented Riemann solver	1D heat equation using an implicit Euler scheme	Jacobi, Laplace and Conjugate Gradient

```
void main() {
   DMR_INIT(user_init_data(), recv_expand());
   DMR_Set_parameter(MIN, MAX, SWEET_SPOT);
   DMR_Inhibit_iter(n_iters);
   for(it = 0; it < ITERATIONS; it++) {
       DMR_RECONFIGURATION( send_expand(),
                            recv_expand(),
                            send_shrink(),
                            recv_shrink());
       compute();
   }
   DMR_FINALIZE(user_free_data());
}
```



Full paper

Bridging the Gap Between Genericity and Programmability of Dynamic Resources in HPC →

Adaptive Resource Management

Graph-based, performance-aware dynamic scheduling

Dynamic Resource Management in HPC systems
using Dynamic Processes with Psets
Dominik Huber et al., HiPC 2025, December 2025

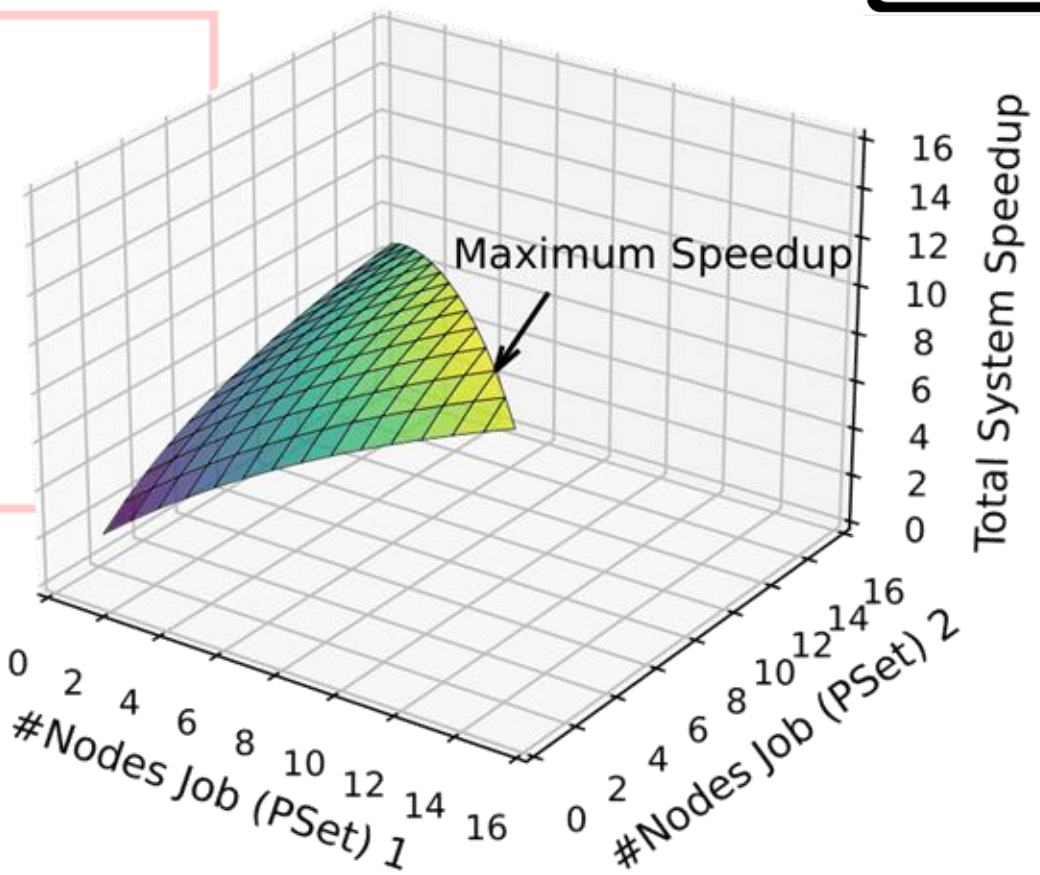
Pre-Print

Objective Function: Global

Global Objective Function for dynamic scheduler.

$$\sum_i w_i \cdot N_{global}(M^i_{psetop}(V^i_{in}, V^i_{out})).$$

N_{global} = Global scalarization
w_i = Global PSetOp wheights



Optimization algorithm used in this work:

Discrete Feasible-Direction Steepest Ascent (DSA)

(over restricted set of feasible exchange moves)

Moving Towards MPI 5.0

Building on the new concepts of MPI 4.0/4.1

- Partitioned Communication, MPI Sessions, Memory kinds
- Increased presence of persistency

New features that are (or IMHO should be) under discussion

- Better support for accelerated architectures
- New language support to enable new communities
- Better integration with task-based runtimes
- Improved tool support
- Malleability
- Fault Tolerance

Fault Tolerance for MPI

Initial approaches failed

- Too heavy weight
- Too much oriented to one model
- …

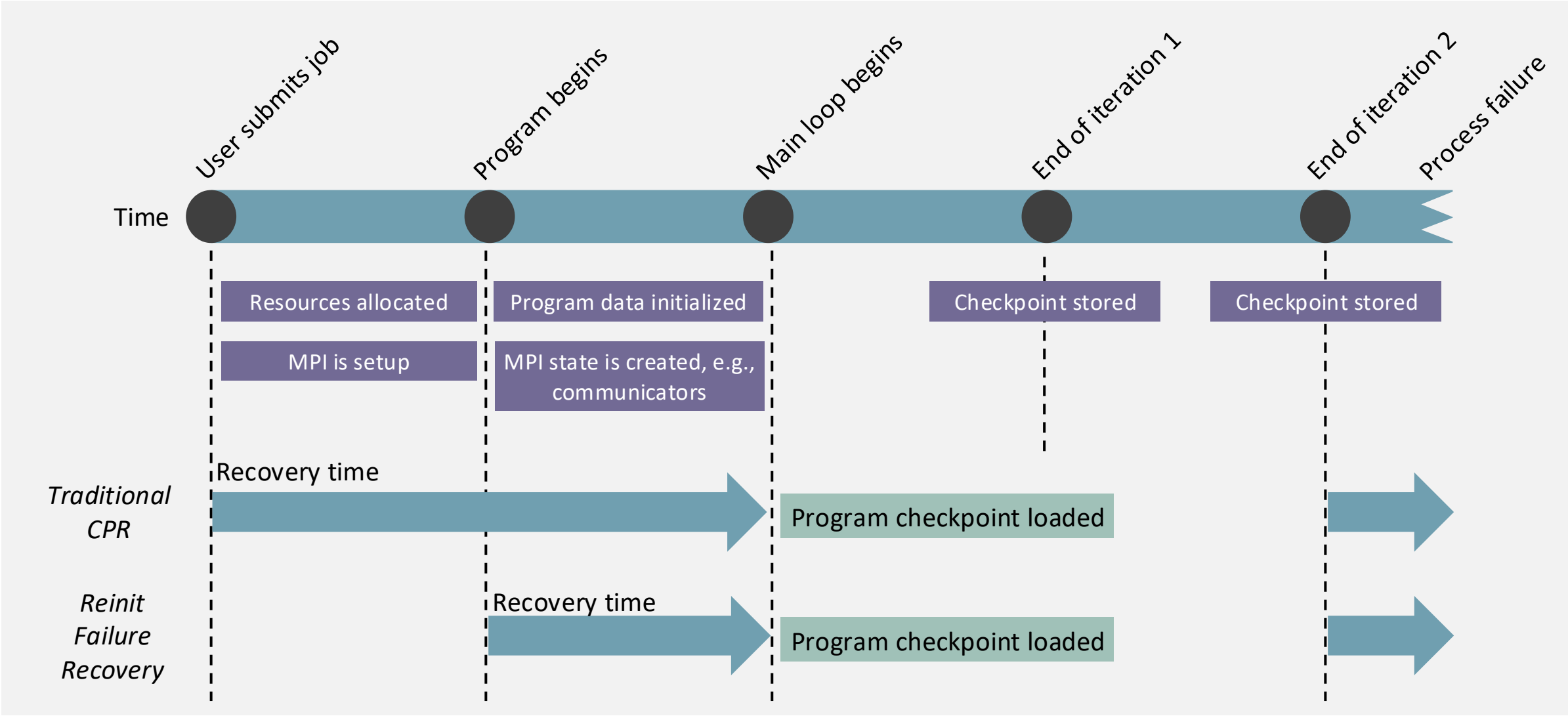
Nevertheless, interest is large

- Continued work in the working group
- Goal: minimal building blocks (many concepts stay, though)
- First step: better error handling in MPI 4.0

Support for several FT models

- Fine grained → ULFM (for new apps)
- Coarse grained → ReInit (to support existing C/R-based apps)
- Session-based
- BUT: still stuck on exact fault mode

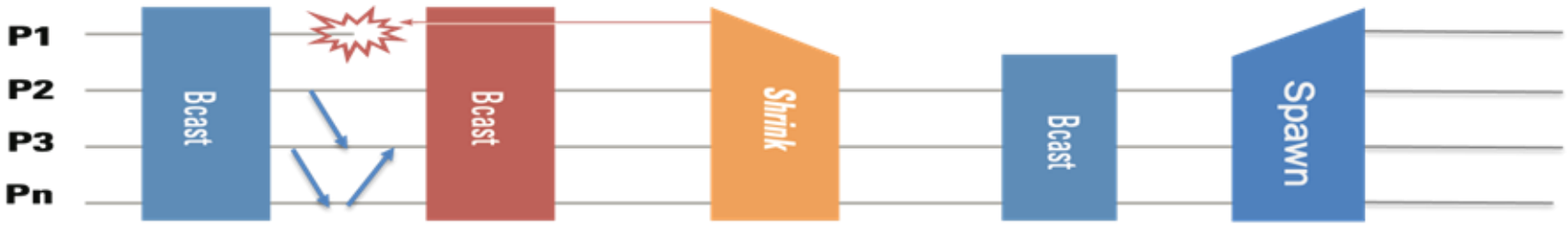
Coarse-grained Recovery (Reinit)



Slide: Ignacio Laguna, LLNL

Fine-grained Fault Tolerance

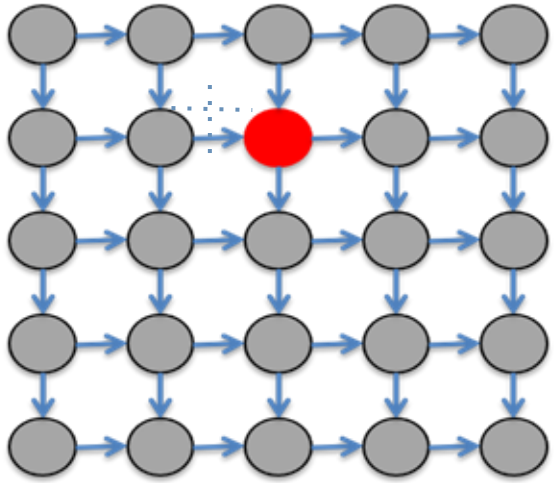
ULFM MPI Crash Recovery (Background)



- Some applications can **continue** w/o recovery
- Some applications are **malleable**
  - Shrink creates a new, smaller communicator on which collectives work
- Some applications are ***not*** **malleable**
  - Spawn can recreate a "same size" communicator
  - It is easy to reorder the ranks according to the original ordering
  - Pre-made code snippets available

- Failure Notification
- Error Propagation
- Error Recovery
- Respawn of nodes
- Dataset restoration

Not all recovery strategies require all of these features, that's why the interface should split notification, propagation and recovery.

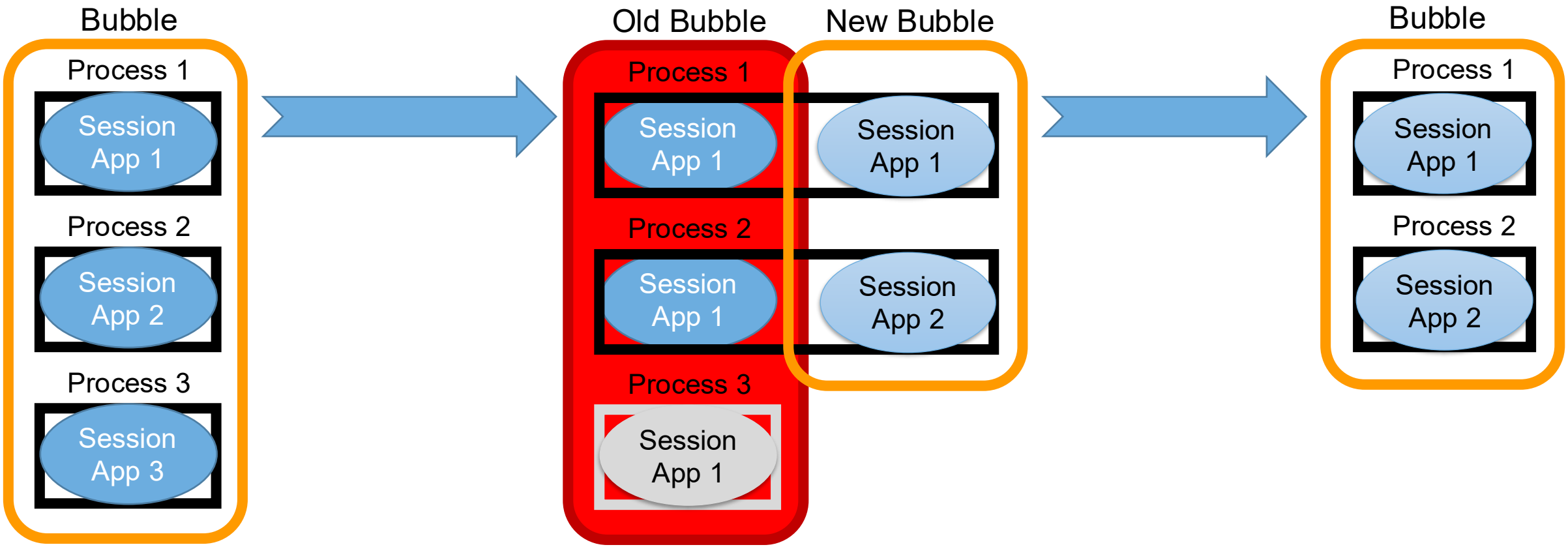


Who should be notified of a failure?
What is the scope of a failure?
What actions should be taken?

- Adds **3 error codes** and **5 functions to manage process crash**
  - **Error codes:** interrupt operations that may block due to process crash
  - MPI_COMM_FAILURE_ACK / GET_ACKED: continued operation with ANY-SOURCE RECV and observation known failures
  - MPI_COMM_REVOKE lets applications interrupt operations on a communicator
  - MPI_COMM_AGREE: synchronize failure knowledge in the application
  - MPI_COMM_SHRINK: create a communicator excluding failed processes
  - More info on the MPI Forum ticket #20: https://github.com/mpi-forum/mpi-issues/issues/20

Slide: Aurelien Bouteiller, UTK

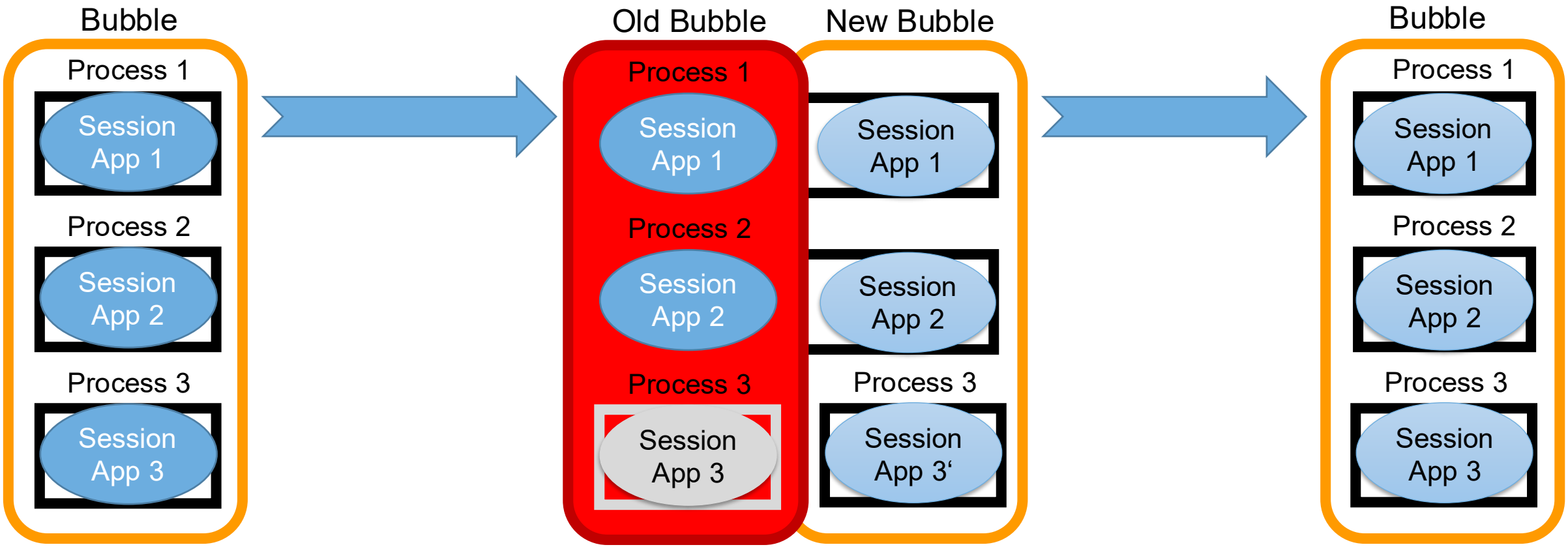
Sessions Can Support Isolation of Failed Processes



Simple support for "Shrinking Recovery"

- Needs functionality to "pop" a bubble
- Supports cleanup, as resources are properly isolated

Sessions Can Support Restart of Resources



Simpe support for “Non-Shrinking Recovery”

- Preferred/required by most applications → default case
- Needs negotiation interface with runtime

The MPI Standard and the MPI Forum

Where we are?

- MPI 4.0 provides powerful new features
- MPI 4.1 offers cleanup and adds minor features
- MPI 5.0 introduces a first ABI

What is next?

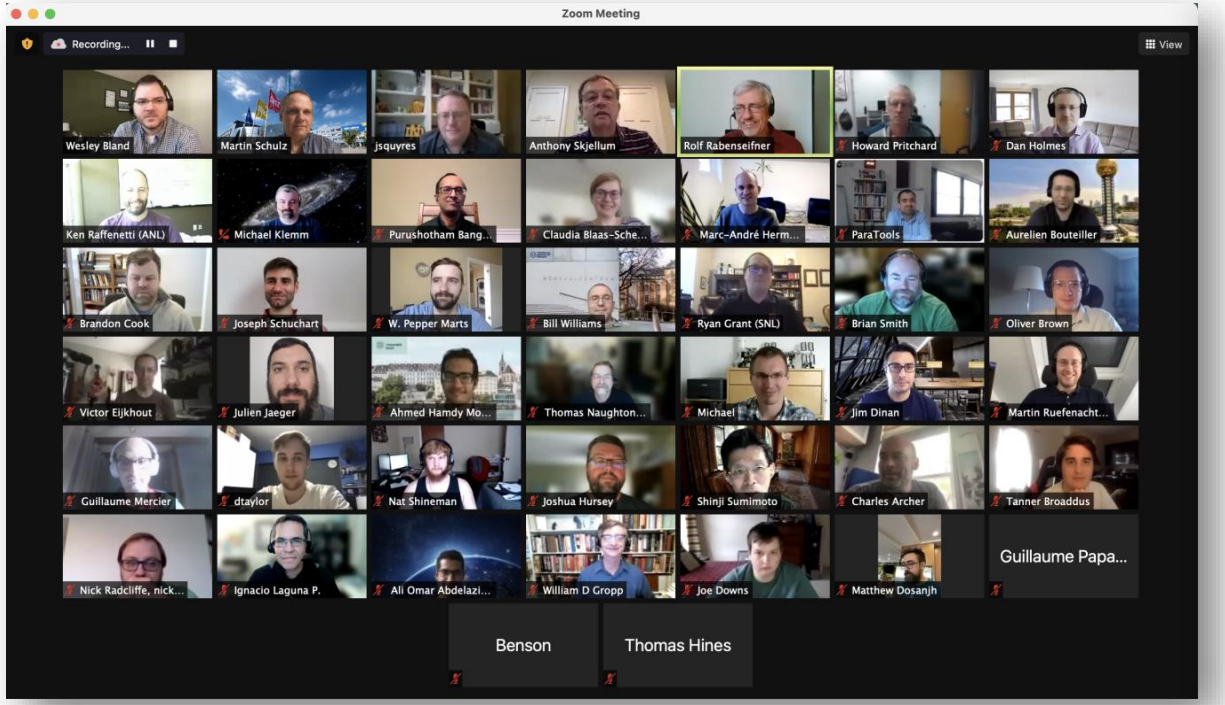
- Many research topics on optimizing new features
- New concepts in MPI 4.1 drive new innovations
  - New support for accelerated systems
  - New language and tooling support
  - Drive towards malleability
- Many more open issues

How can you participate?

- The MPI Forum is an open organization
- Join a WG, participate in meetings
- Introduce new concepts so all can benefit

It takes a team, or rather teams …

The TUM CAPS Team



The MPI Forum

Bayerisches Staatsministerium für Wissenschaft und Kunst

Bayerische Forschungsstiftung

SPONSORED BY THE

Federal Ministry of Education and Research

Federal Ministry for Economic Affairs and Climate Action

The MPI Standard and the MPI Forum

Where we are?

- MPI 4.0 provides powerful new features
- MPI 4.1 offers cleanup and adds minor features
- MPI 5.0 introduces a first ABI

What is next?

- Many research topics on optimizing new features
- New concepts in MPI 4.1 drive new innovations
  - New support for accelerated systems
  - New language and tooling support
  - Drive towards malleability
- Many more open issues

How can you participate?

- The MPI Forum is an open organization
- Join a WG, participate in meetings
- Introduce new concepts so all can benefit

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