150 Jahre
culture of
excellence

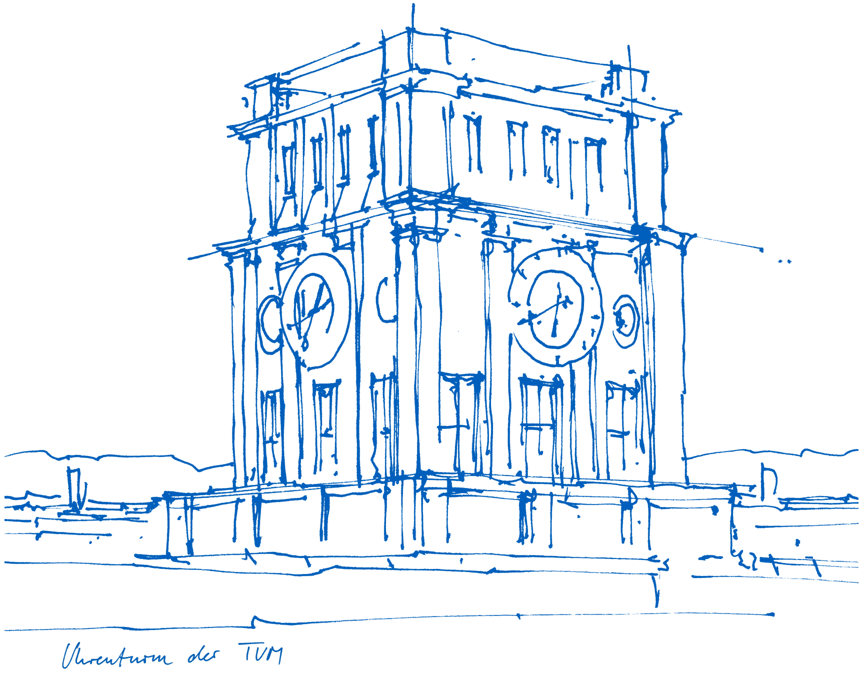
Extending and Updating the Tool Interfaces in MPI: A Request for Feedback

Martin Schulz
Technische Universität München
Fakultät für Informatik

Scalable Tools Workshop 2018
Solitude, UT
July 2018

With material from

- Marc-Andre Hermanns, JSC
- Kathryn Mohror, LLNL



Tools Activities in the MPI Forum

Tools WG – Leads:

- Marc-Andre Hermanns, JSC
- Kathryn Mohror, LLNL

Focuses on all aspects of tool interfaces in MPI

- Debugging and performance tools
- Impact on other parts of the standard

Currently under discussion

- **MPI_T Events – adding callbacks to MPI_T**
- **QMPI – modernizing PMPI**
- UUIDs for variables and events – easier identification and tracking
- Timers – integers instead of doubles
- Debug interface vs. PMI / PMIx
- What do “Sessions” mean for tools?

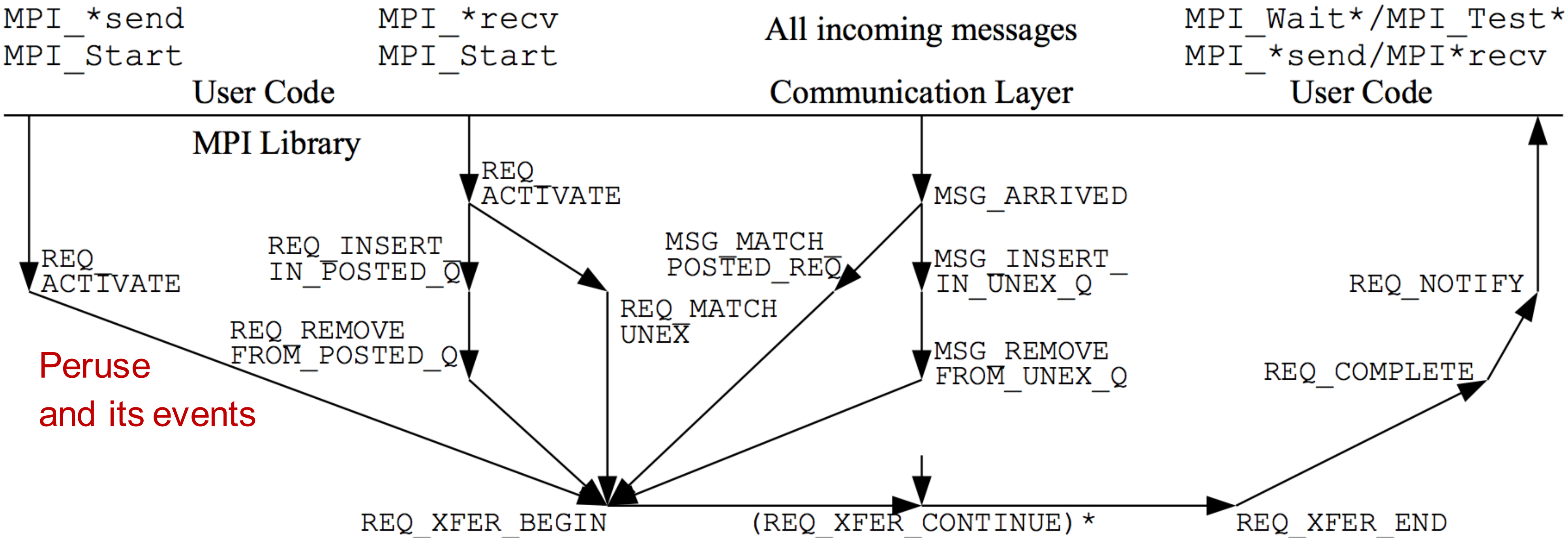
Part 1: MPI_T Events

Motivation

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

Didn‘t we see the idea of MPI events for tools before? Yes: MPI Peruse

- Access to specific runtime events
- List of point-to-point events defined
- Prototyped, but never standardized



Peruse and its events

MPI_T Events Builds on the Ideas of MPI_T

Do not mandate specific implementation of MPI functionality

- No requirement to implement specific events

Provide access to MPI implementation-internal information about events

- What happens and when it happens

Notification of events can be immediate or deferred

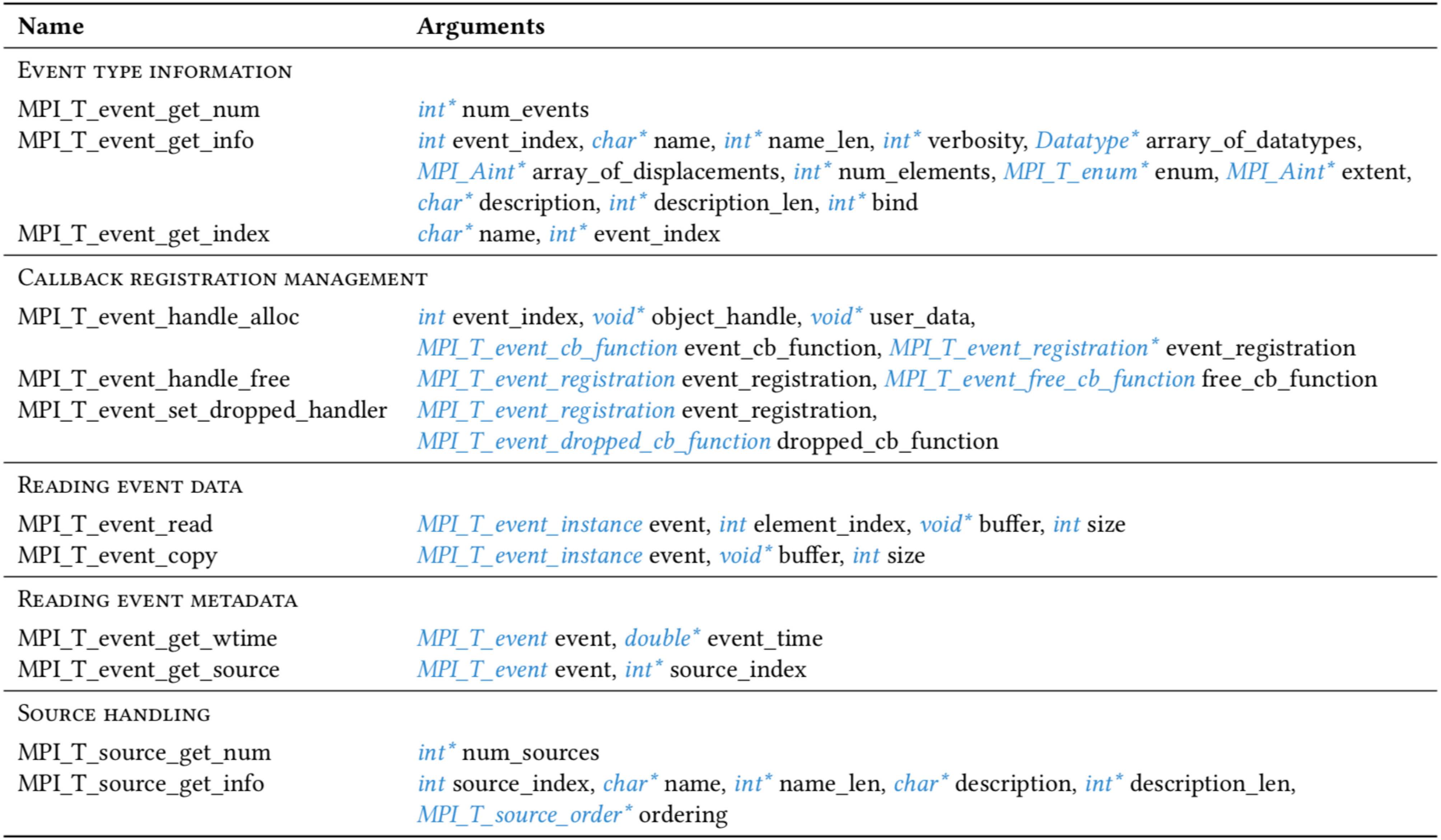
- Queuing of events can reduce overhead
- It may be impossible to provide immediate notification of some events

Matches the concepts of the existing MPI_T interface

- Interface

A) to query available events	(query variables)
B) register callbacks	(allocate handles)
C) read data during callbacks	(read variables)

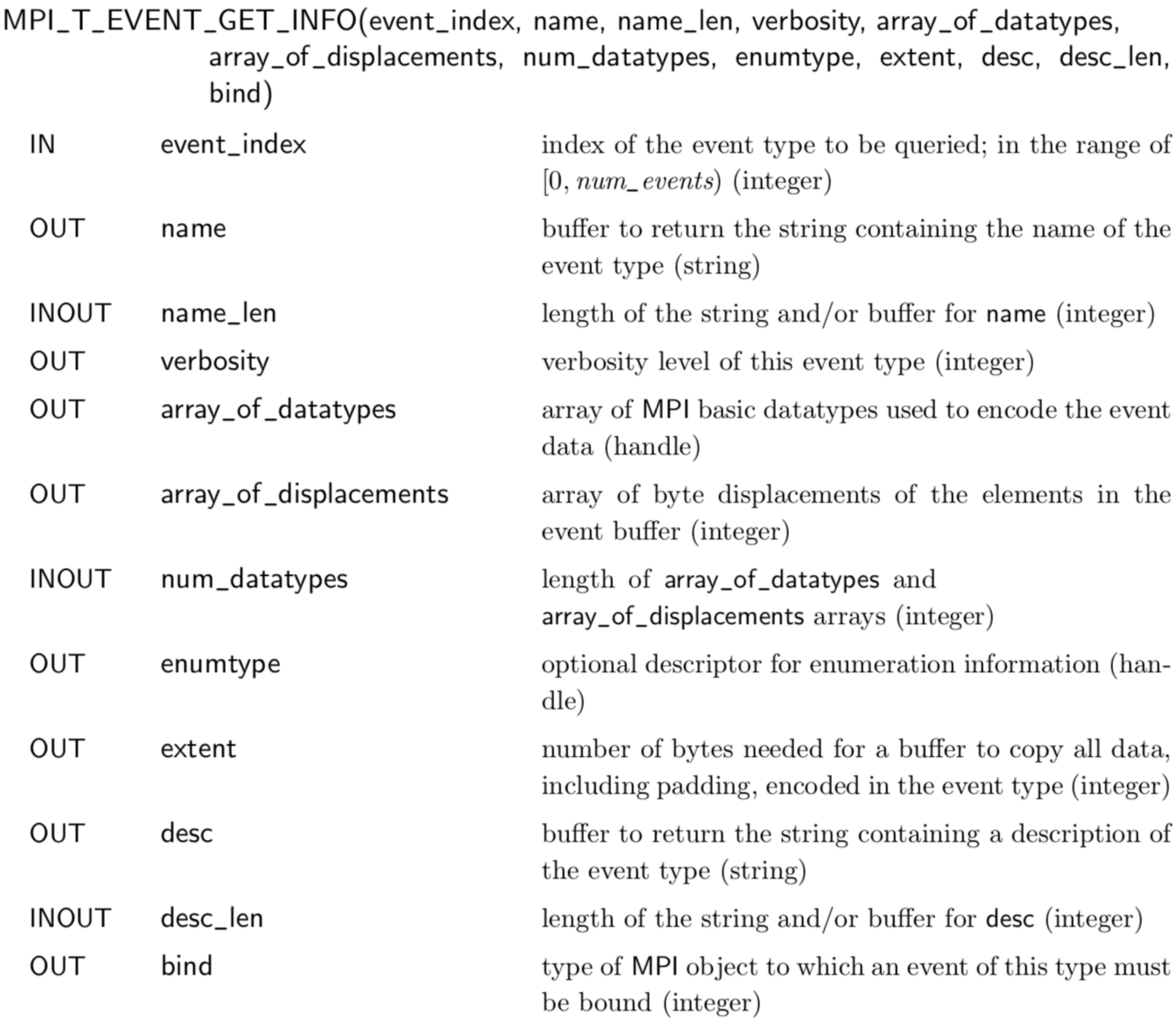
Complete MPI_T Events API



Name	Arguments
EVENT TYPE INFORMATION	
MPI_T_event_get_num	int* num_events
MPI_T_event_get_info	int event_index, char* name, int* name_len, int* verbosity, Datatype* arrary_of_datatypes, MPI_Aint* array_of_displacements, int* num_elements, MPI_T_enum* enum, MPI_Aint* extent, char* description, int* description_len, int* bind
MPI_T_event_get_index	char* name, int* event_index
CALLBACK REGISTRATION MANAGEMENT	
MPI_T_event_handle_alloc	int event_index, void* object_handle, void* user_data, MPI_T_event_cb_function event_cb_function, MPI_T_event_registration* event_registration
MPI_T_event_handle_free	MPI_T_event_registration event_registration, MPI_T_event_free_cb_function free_cb_function
MPI_T_event_set_dropped_handler	MPI_T_event_registration event_registration, MPI_T_event_dropped_cb_function dropped_cb_function
READING EVENT DATA	
MPI_T_event_read	MPI_T_event_instance event, int element_index, void* buffer, int size
MPI_T_event_copy	MPI_T_event_instance event, void* buffer, int size
READING EVENT METADATA	
MPI_T_event_get_wtime	MPI_T_event event, double* event_time
MPI_T_event_get_source	MPI_T_event event, int* source_index
SOURCE HANDLING	
MPI_T_source_get_num	int* num_sources
MPI_T_source_get_info	int source_index, char* name, int* name_len, char* description, int* description_len, MPI_T_source_order* ordering

Query API

Query available events and their semantic info

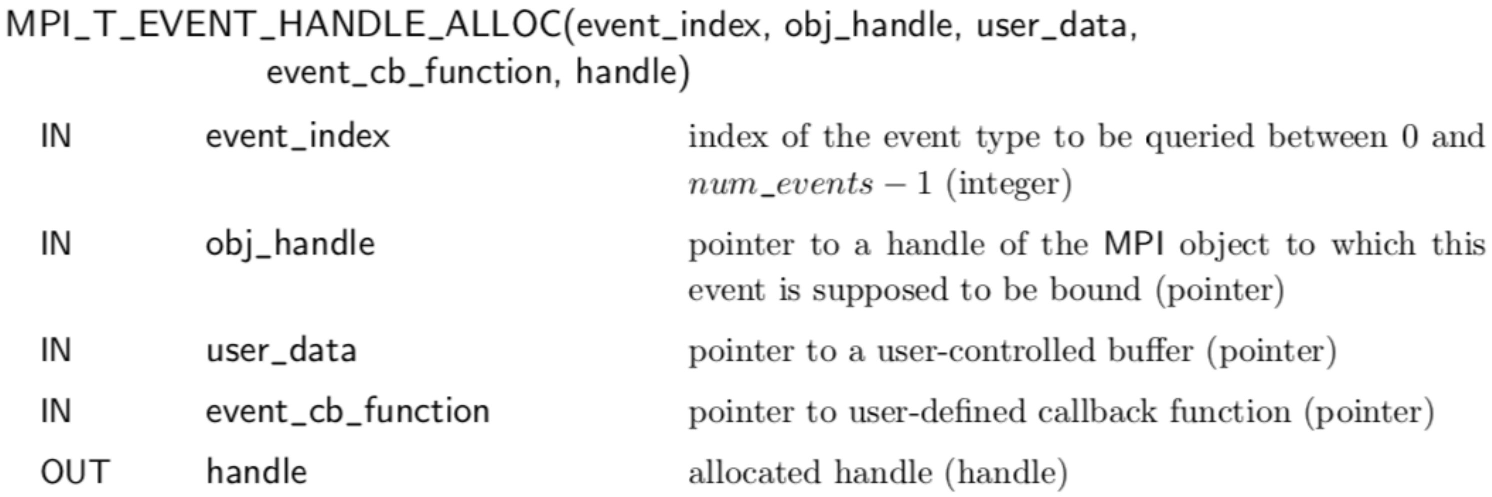


MPI_T_EVENT_GET_INFO(event_index, name, name_len, verbosity, array_of_datatypes, array_of_displacements, num_datatypes, enumtype, extent, desc, desc_len, bind)

IN	event_index	index of the event type to be queried; in the range of $[0, num\_events)$ (integer)
OUT	name	buffer to return the string containing the name of the event type (string)
INOUT	name_len	length of the string and/or buffer for name (integer)
OUT	verbosity	verbosity level of this event type (integer)
OUT	array_of_datatypes	array of MPI basic datatypes used to encode the event data (handle)
OUT	array_of_displacements	array of byte displacements of the elements in the event buffer (integer)
INOUT	num_datatypes	length of array_of_datatypes and array_of_displacements arrays (integer)
OUT	enumtype	optional descriptor for enumeration information (handle)
OUT	extent	number of bytes needed for a buffer to copy all data, including padding, encoded in the event type (integer)
OUT	desc	buffer to return the string containing a description of the event type (string)
INOUT	desc_len	length of the string and/or buffer for desc (integer)
OUT	bind	type of MPI object to which an event of this type must be bound (integer)

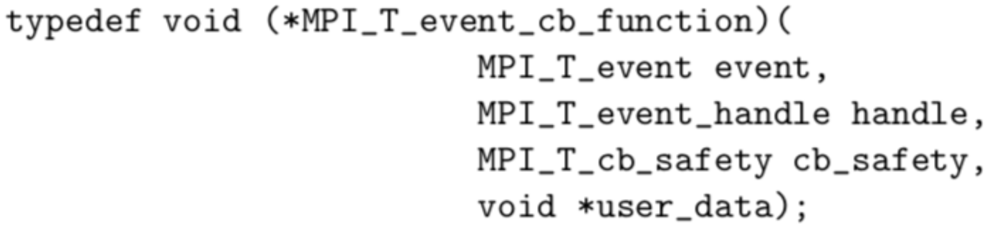
Allocating Event Handles and their Callbacks

Register for events of interest:



MPI_T_EVENT_HANDLE_ALLOC(event_index, obj_handle, user_data,
event_cb_function, handle)

IN	event_index	index of the event type to be queried between 0 and *num_events* − 1 (integer)
IN	obj_handle	pointer to a handle of the MPI object to which this event is supposed to be bound (pointer)
IN	user_data	pointer to a user-controlled buffer (pointer)
IN	event_cb_function	pointer to user-defined callback function (pointer)
OUT	handle	allocated handle (handle)

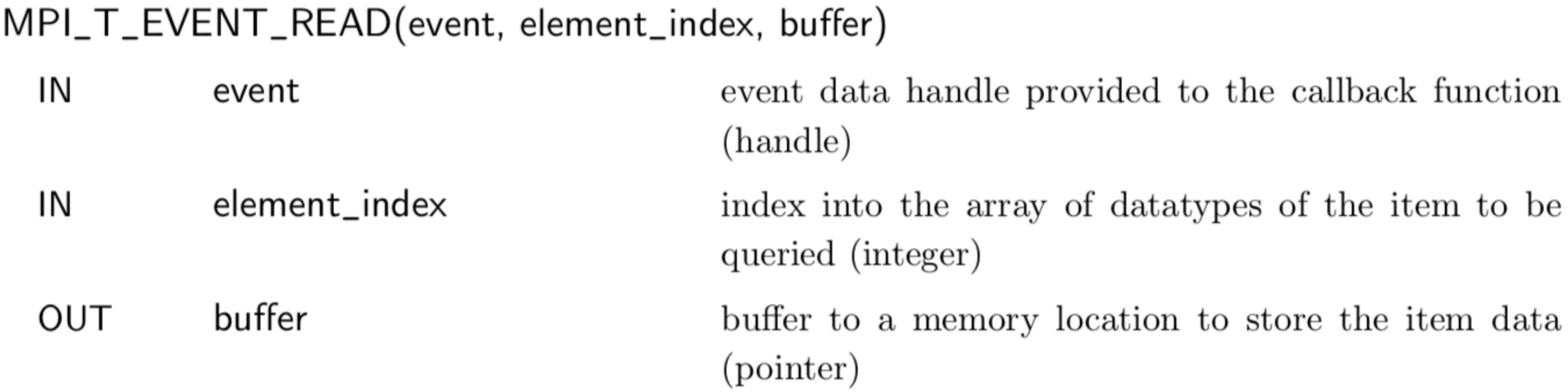


```
typedef void (*MPI_T_event_cb_function)(
                        MPI_T_event event,
                        MPI_T_event_handle handle,
                        MPI_T_cb_safety cb_safety,
                        void *user_data);
```

Receiving Callbacks

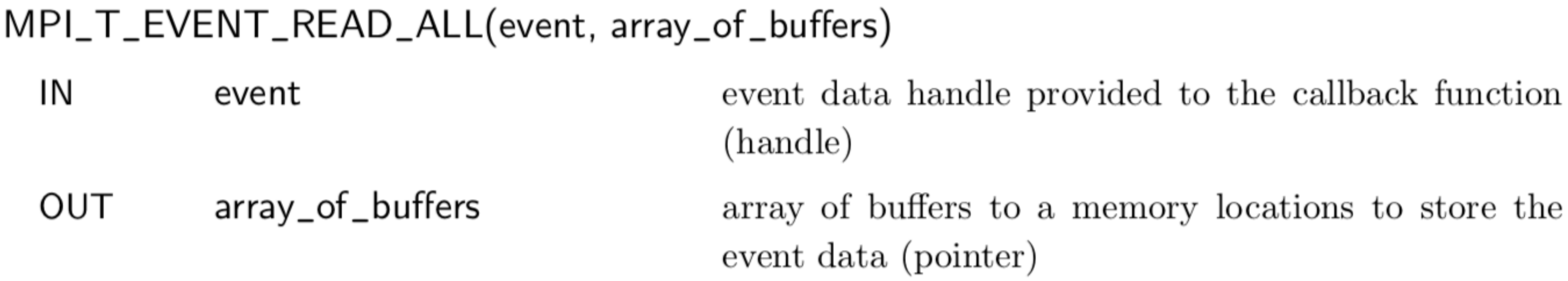
Callbacks for allocated handles are triggered when the corresponding event happens

- Opaque MPI_T event type can be queried for information
- Type scheme still under discussion



MPI_T_EVENT_READ(event, element_index, buffer)

IN	event	event data handle provided to the callback function (handle)
IN	element_index	index into the array of datatypes of the item to be queried (integer)
OUT	buffer	buffer to a memory location to store the item data (pointer)



MPI_T_EVENT_READ_ALL(event, array_of_buffers)

IN	event	event data handle provided to the callback function (handle)
OUT	array_of_buffers	array of buffers to a memory locations to store the event data (pointer)

Special Provisions

Handling of calling safety for callbacks

- Only minimal MPI usage allowed
- Each callback can state the “safety level” at each event instance
- None, Reentrant, thread safe, async signal safe

MPI_T Events implementations allowed to defer events

- Provide timestamps to match up deferred events

MPI_T Events implementations allowed to drop events

- Should be the exception, but can be necessary
- Special dropped event handler to indicate dropping to tool

Ordering of events

- Concept of event sources
- Events from the same source are ordered
- Events from different sources can be out of order

Status: MPI_T Events

Proposal mostly complete

- https://github.com/mpiwg-tools/tools-issues/wiki/MPI_T-Events
- Current proposal text available on request
- “Reading” planned for September MPI Forum meeting

Prototype implementation close to being done

- Based on Open MPI
- Providing Peruse functionality

Publication

- Enabling callback-driven runtime introspection via MPI_T
  Hermanns, Hjelm, Knobloch, Mohror, Schulz
  To appear in EuroMPI 2018

Part 2: QMPI

Proposal to redesign the trusted PMPI interface

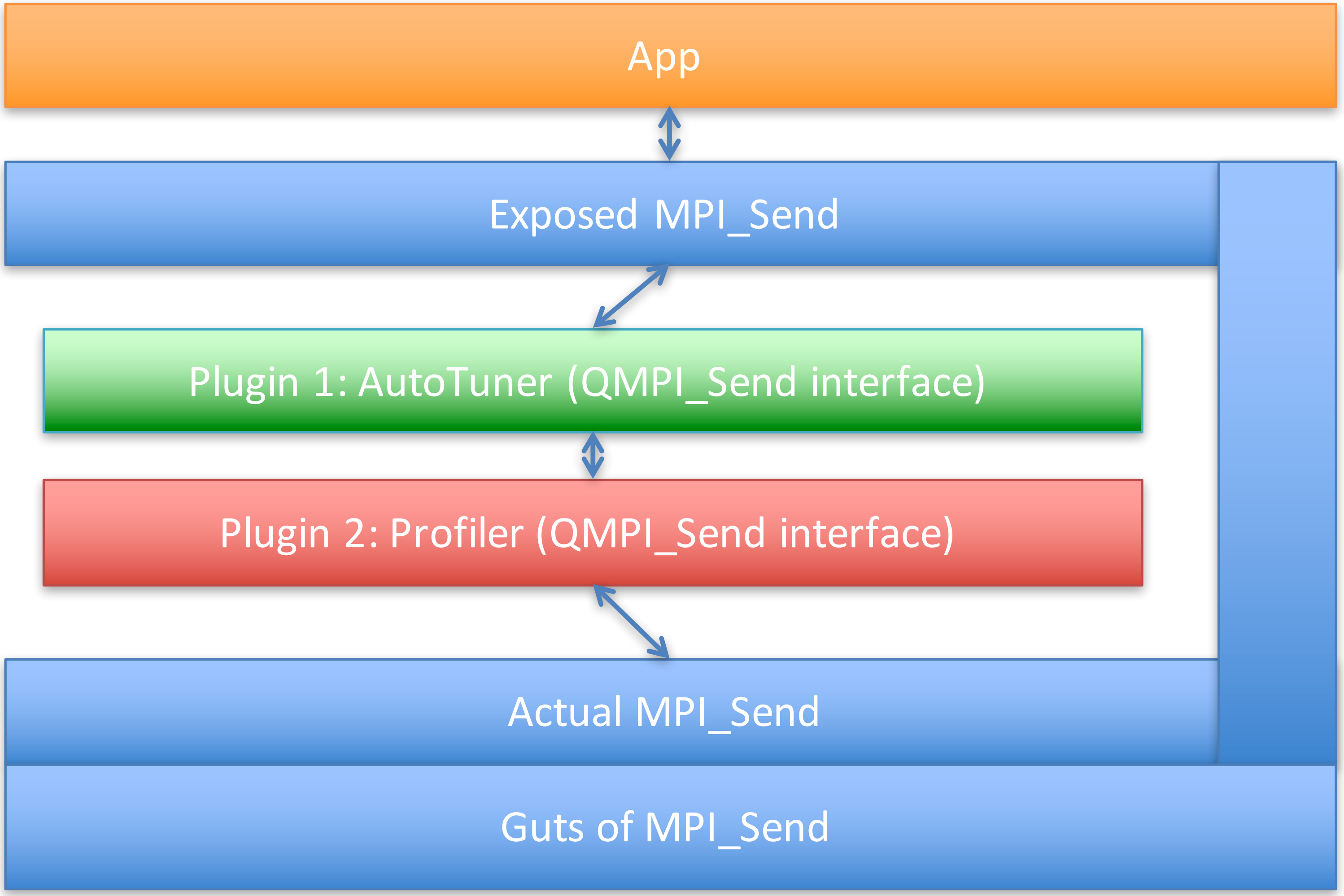
Motivation

- Weak symbol intersection is brittle
- Limited to a single tool (unless you use the awesome P^{n}MPI)
- Forces tools to be monolithic

Requirements

- Support multiple concurrent tools in a single process
- Link time or runtime enablement
- Low to no overhead when no tool is attached
- No loss of functionality compared to existing PMPI
  - Basically wrapper functionality
- All language bindings (C, mpif.h, use mpi, use mpif08)
  - Tools can implement functionality in C (in one place) regardless of language
- Integration with MPI thread support

Basic Scenario Targeted at First



App

Exposed MPI_Send

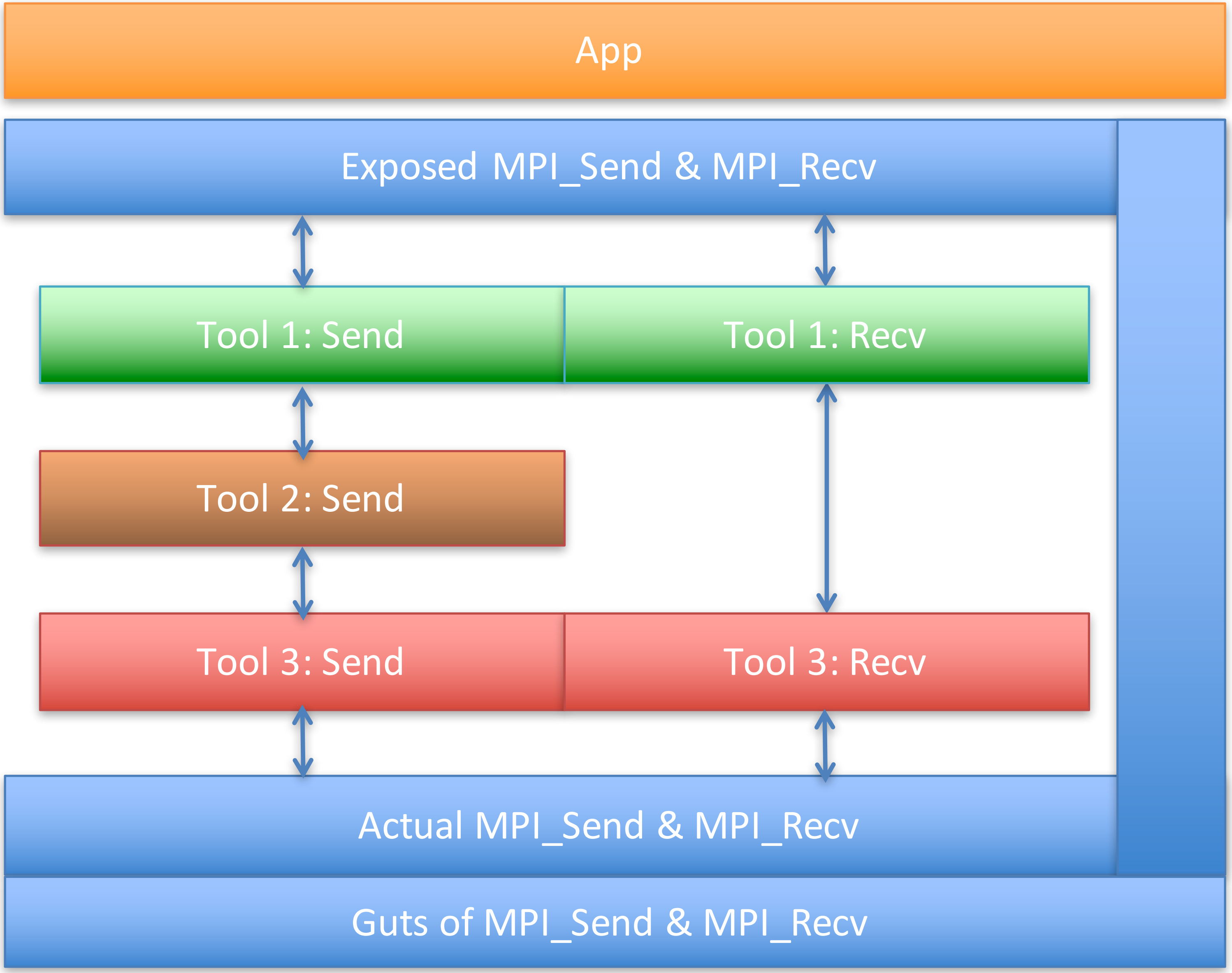
Plugin 1: AutoTuner (QMPI_Send interface)

Plugin 2: Profiler (QMPI_Send interface)

Actual MPI_Send

Guts of MPI_Send

Basic Scenario Targeted at First


App
Exposed MPI_Send & MPI_Recv
Tool 1: Send
Tool 1: Recv
Tool 2: Send
Tool 3: Send
Tool 3: Recv
Actual MPI_Send & MPI_Recv
Guts of MPI_Send & MPI_Recv

Basis is Still Basic Wrapping

Each tool implements a set of routines it wraps

- Registered at startup

Tools have independent instances

- Separate storage space
- Created by MPI at/before MPI Event

Each tool instance has the following “available”:

- A functional table with all “PMPI” / follow on routines
- A pointer to store internal information

```
Wrapping process:
   Int QMPI_X( <normal parameters>, opaque)
   {
         qmpi_x_t pqmpi_x;
         MPI_Table_query(“QMPI_X”, &pqmpi_x, table);
         ... Do work ...
         err=pqmpi_x(..., opaque);
         ... Do work ...
         return err;
   }
```

Status: QMPI

Concept mostly worked out

- https://github.com/mpiwg-tools/tools-issues/wiki/Interface-to-Replace-PMPI
- APIs are being defined
- Working on standards text is coming up soon-ish

Active work on

- Initialization / Bootstrapping
- Opaque information passed through
- Ability to clean ”loop back” to own layer

Prototype implementation in the works

- As PMPI tool that provides the new interface
- Basic wrapping already possible
- Generalization of the next few months

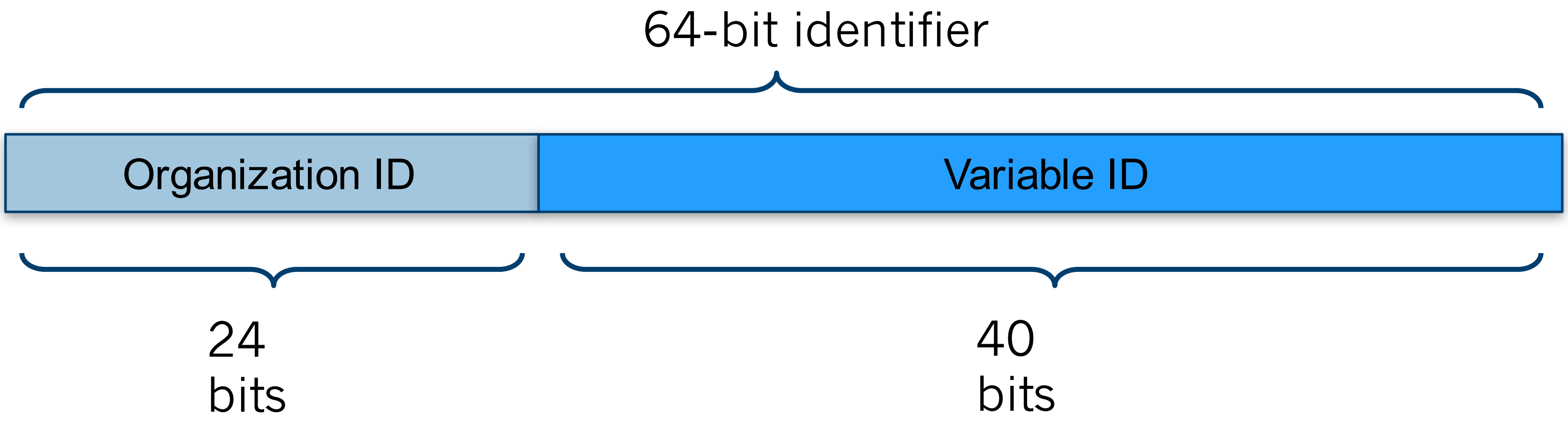
Part 3: UUIDs for MPI_T variables

MPI implementations are free to provide whatever variables make sense for their implementation

- Variables are allowed to change between versions of the library and across HW (analog to performance and control variables)
- Want to provide some stability for tools and keep the freedom for implementations

Organization IDs and variable identifiers registered with MPI Forum

- Allows to identify common variables across MPI implementations
- Allows to keep variables across MPI versions uniquely identifiable



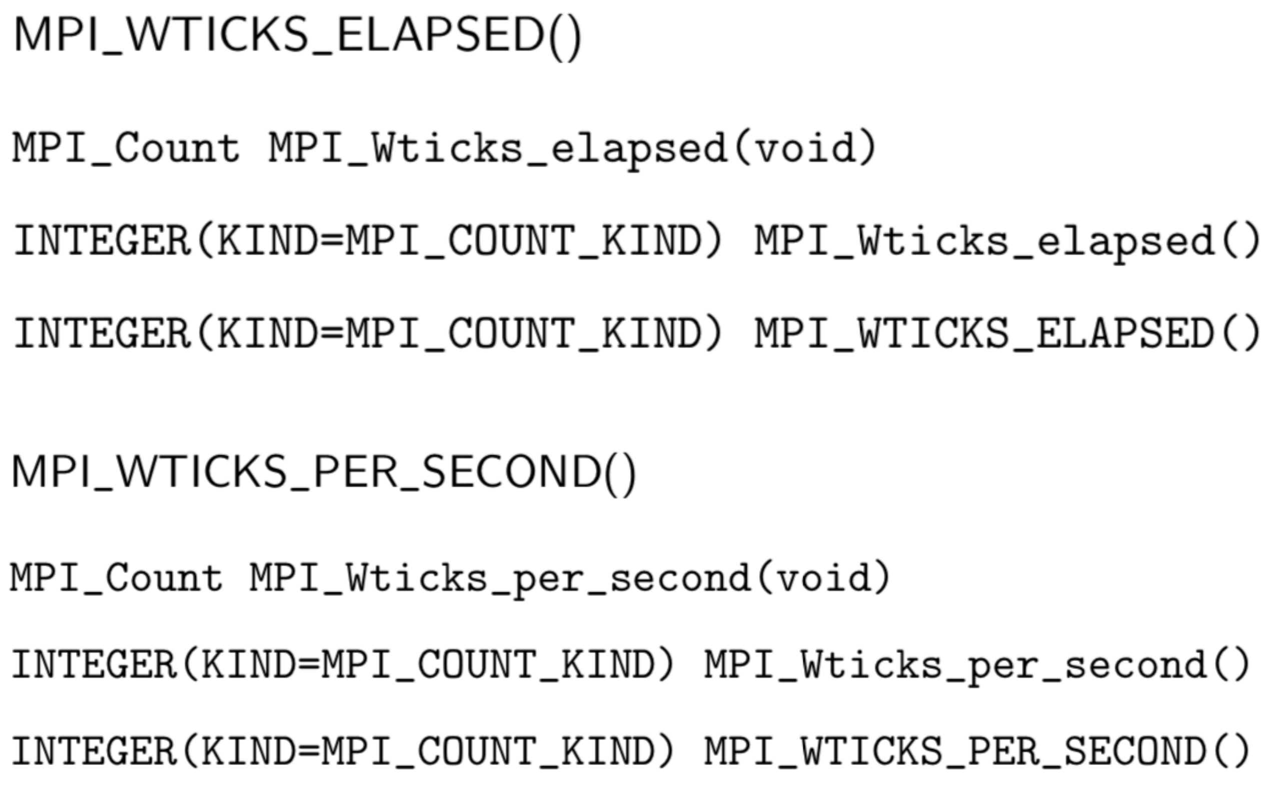
Vendors are allowed to _use_ a "foreign" VendorID for a variable that has the same semantics as the corresponding variable

Part 4: Timers

Issue 1: Timers only provide double, which requires conversions for some sources

Proposal 1: new general timing routines

Proposal 2: new MPI_T timers, possibly per source (currently preferred)



MPI_WTICKS_ELAPSED()

```
MPI_Count MPI_Wticks_elapsed(void)

INTEGER(KIND=MPI_COUNT_KIND) MPI_Wticks_elapsed()

INTEGER(KIND=MPI_COUNT_KIND) MPI_WTICKS_ELAPSED()
```

MPI_WTICKS_PER_SECOND()

```
MPI_Count MPI_Wticks_per_second(void)

INTEGER(KIND=MPI_COUNT_KIND) MPI_Wticks_per_second()

INTEGER(KIND=MPI_COUNT_KIND) MPI_WTICKS_PER_SECOND()
```

Issue 2: MPI timing routines cannot be called before MPI_Init

Proposal: ???

Summary and Request for Feedback

Currently under discussion

- **MPI_T Events – adding callbacks to MPI_T**
- **QMPI – modernizing PMPI**
- UUIDs for variables and events – easier identification and tracking
- Timers – integers instead of doubles
- Debug interface vs. PMI / PMIx
- What do “Sessions” mean for tools?

If you have feedback, please send it to

- Marc-Andre: m.a.hermanns@fz-juelich.de
- Kathryn Mohror: mohror1@llnl.gov
- Martin Schulz: schulzm@in.tum.de

Or join the WG

- TelCons: Thursday at 8am Pacific Time | 5pm CET
- More Information on Github:
- https://github.com/mpiwg-tools/tools-issues