Use Cases: MPIT from the View of Tool Developers

- Number of times one hit an “inefficient path”
  - Potentially caused by resource limitations
  - Basically a set of counters on certain execution paths
- Resource exhaustion (where? Which resource?)
- Queue length
- Memory footprint and allocation reasons
- Time used for matching messages
  - Can also be use case for piggybacking
- Time spent between entering data and sending data
  - Especially for collectives
- High watermarks for such timings
Technical Details (1): Extensibility

- Extensibility wanted several groups
  - Add new variables through additional instrumentation
- Tool writers want PMPIT interception
- Mechanisms to allow adding of variables (assuming PMPIT)
  - Routines to allocate dummy handles
  - Change iterators to be based on integers
  - Don’t allow MPIT to remove/renumber variables
- Return structs with query information
  - Easier extensibility
  - Guarded by a separate MPIT version (#define)
Technical Details (2): Setting Control Variables

- Generally seen as useful and should be part of the proposal
  - But could use more use cases with concrete numbers

- Add fields to query information
  - Readonly: can never be set
  - Sync/Nosync: does setting require a global operation
  - Comm: Communicator scope

- Change semantics of set routine
  - Arguments: name, value, communicator
  - Pass in communicator from above
  - Must be called by all member of the communicator
  - Can be MPI_COMM_SELF (local)
Technical Details (3): Other Changes

- **Renaming**
  - Configuration Variables -> Control Variables
  - MPIT_CTRLVARS…
  - MPIT_PERFVARS…

- **Allow 1-N for verbosity levels (+ call for max. verbosity)**

- **Initialization**
  - Needs some more discussion
  - Remove IsInitialized and IsFinalized calls

- **Bias for adding Fortran bindings**
  - Need to make sure API still works
  - Needs more feedback