PMIx unit-testing

Artem Y. Polyakov
NVIDIA

Charles Shereda
LANL

Boris I. Karasev
NVIDIA

Howard Prichard
LANL

Importance of the testing problem

More and more HPC (and not only HPC) software starts to rely heavily on PMIx ...

Resource Managers supporting PMIx

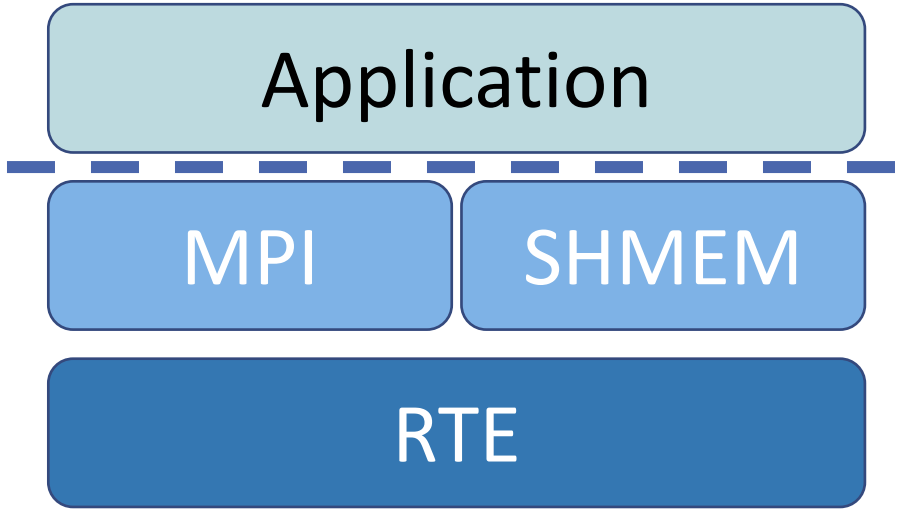
- Slurm
- IBM Job Step Manager (JSM)
- Fujitsu Job Scheduler
- etc.

MPI/OSHMEM implementations

- Open MPI/OpenSHMEM will fully rely on PMIx starting from v5.0
  - => derivatives: IBM Spectrum MPI, NVIDIA/Mellanox HPCX, Fujitsu MPI
- MPICH introduced support for PMIx
  - => derivatives: Intel MPI
- etc.

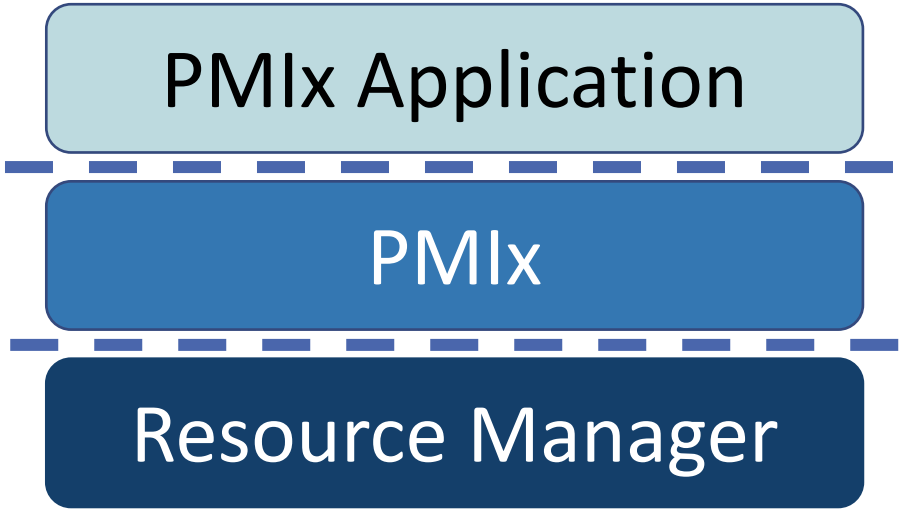
PMIx testing specifics

Most of the HPC APIs



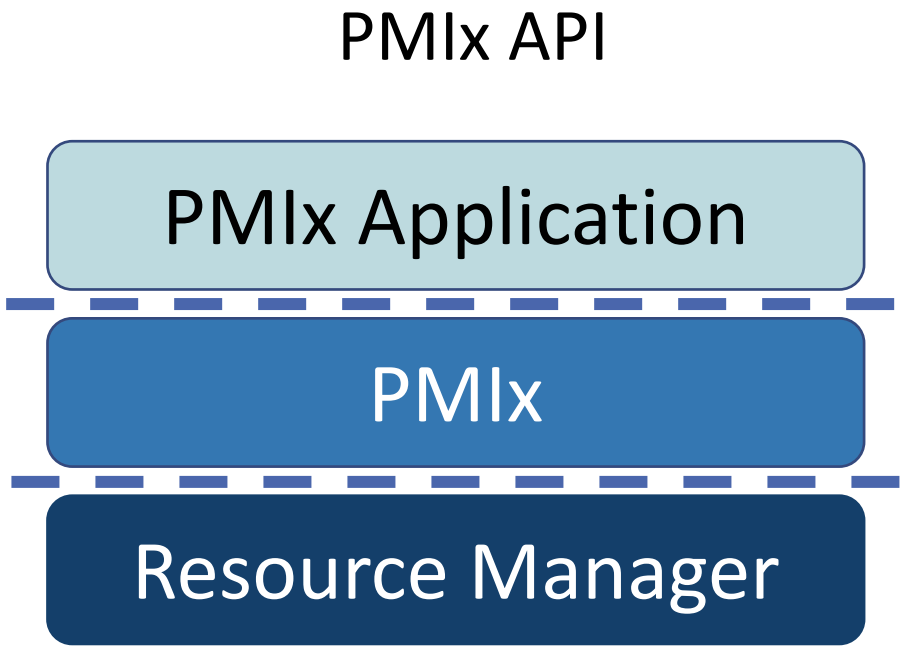
- Interface only user-side
- Do not impose any assumptions on interactions with RTE

PMIx API



- Interface 2 sides
- application-side impact => certain effect on the server-side
- Do not want to debug RM!

PMIx testing specifics (2)

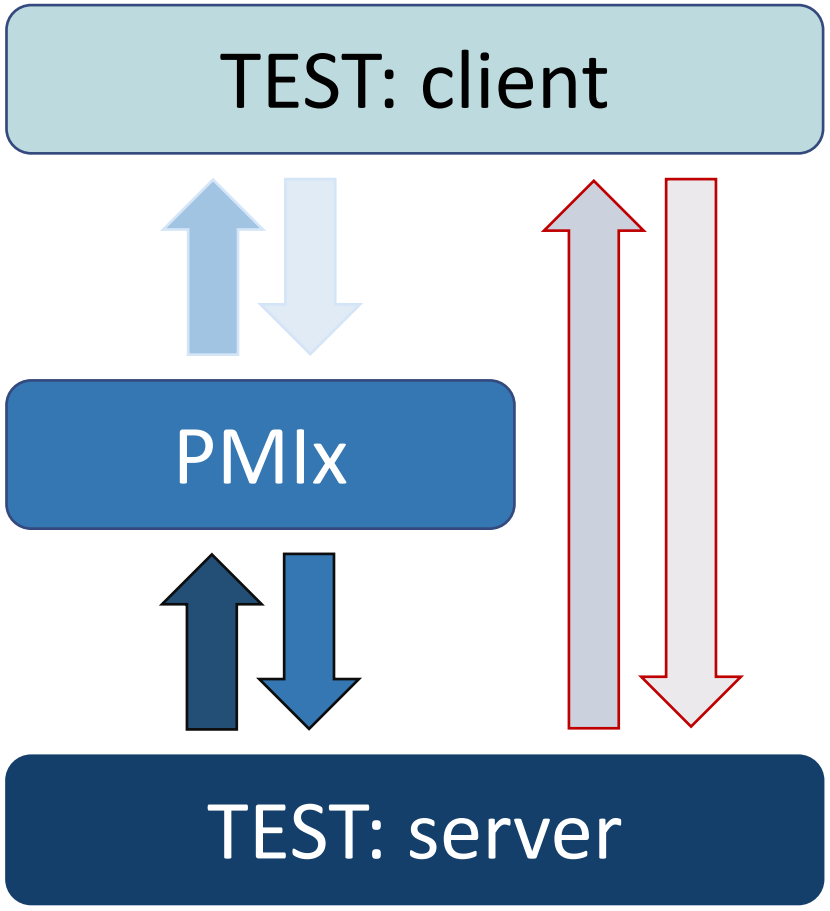


- Interface 2 sides
- application-side impact => certain effect on the server-side

Examples:

Client impact	**Server effect**
PMIx_Init()	N x client_connected()
single-node PMIx_Fence(collect=1)	0 x fence_nb() 0 x direct_modex()
single-node PMIx_Fence(collect=0)	0 x fence_nb() 0 x direct_modex()
multi-node PMIx_Fence(collect=1)	1 x fence_nb() 0 x direct_modex()
multi-node PMIx_Fence(collect=0)	1 x fence_nb(size=0) M x direct_modex()
PMIx_Get(immediate)	0 x direct_modex()

Desired testing setup



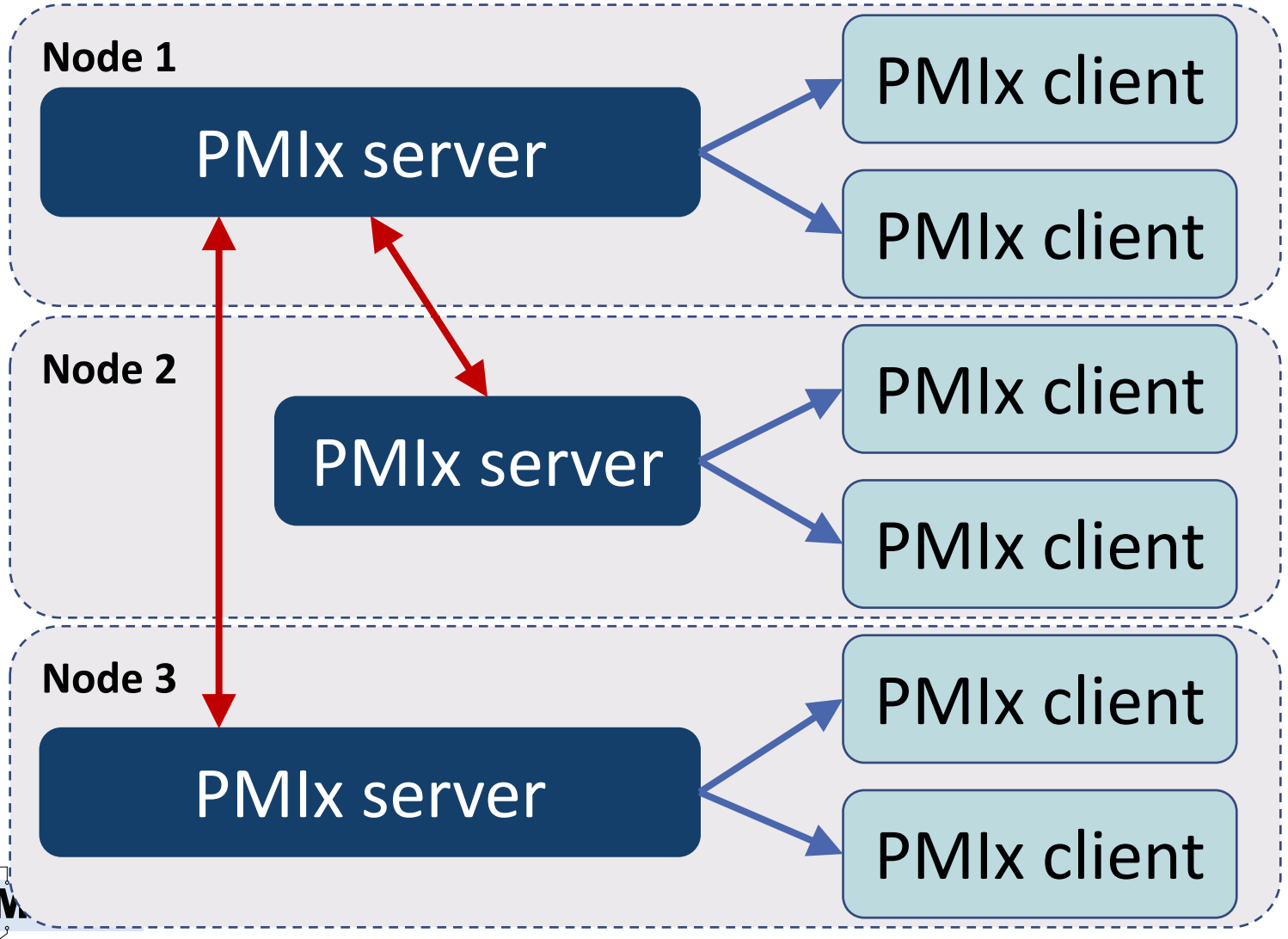
PMIx server-side API

PMIx client-side API

TEST: side-channel

Existing test-suite

```
$ pmix_test -n 6 -s 3 [ test-options ] ./pmix_client
```



POSIX pipes

PMIx API

- **Side-channel=[test-options]**
  - uni-directional
  - static, not flexible
- **Tests are old (2015)**
  PMIx semantics was evolving
  Ex.: Dmdx => Fence is [not] req'd
- **Tests are complex**
  testing many things at once
- **Single-binary / Many-test**

New design goals

- Fine-grained unit-tests
  - Test small functionality subset
  - Well-defined, discussed and well-understood by all developers
    - Each test goes through a **formal approval procedure**
  - Incrementally test new functionality relying on previous tests
  - Execute each test multiple times to uncover race conditions
  - Extend the test suite based on bug reports analysis
- Simple reliable bi-directional side-channel:
  - Server-side communicates static job-level information
  - Client-side interactively communicates the expected effect
    - When the server-side effect is expected on the applied impact

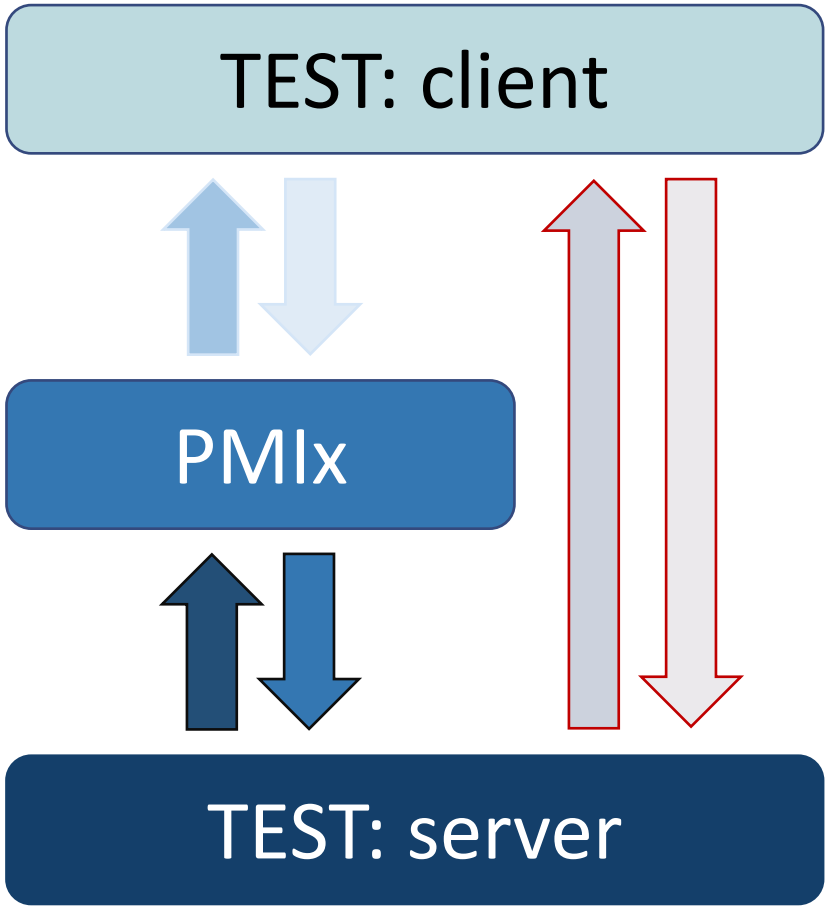
New test formal approval procedure

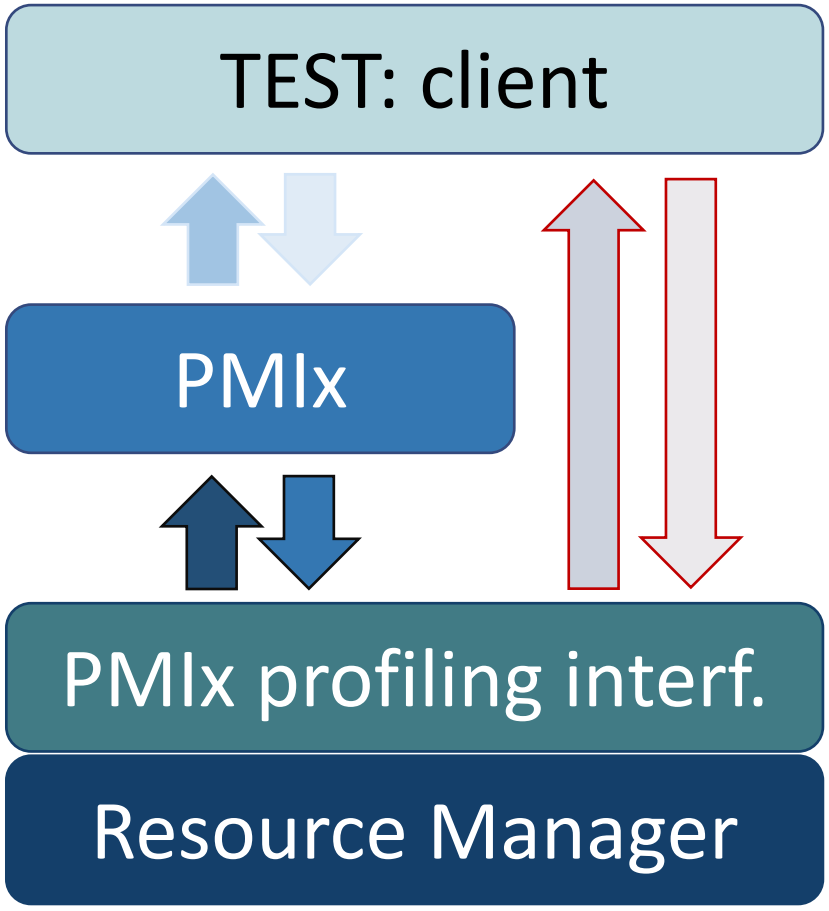
- A issue is created in **openpmix/pmix-tests** repository on GitHub
  - The issue message includes description of the test that includes at least the following required items:
    - Test description
    - Client-side expectations
    - Server-side expectations
    - Reference implementation
- In parallel, in **openpmix/openpmix** repository, a reference implementation is provided in a Pull Request
- All semantics related discussions are preserved in the issue
- The Issue message is updated as needed to reflect the ultimate test description
- Once test is approved by all interested parties, the corresponding PR is merged and the issue is closed and can later be used for the reference.

Current state of the implementation

- **Codebase**
  - The implementation leverages previously developed test suite to implement the new design goals.
- **Side-channel:**
  - [DONE] Static (job-/node-/app-/rank-level) information is passed through a base64-encoded serialized structure to maintain flexibility and simplify implementation.
  - [WIP] Client-side interactive communication to allow the test to drive the server side.
- **Tests:**
  - **PMIx hello-world**: Passed Formal approval
  - **PMIx job info/rank positioning**:
    - Issue: https://github.com/openpmix/pmix-tests/issues/64
    - PR: WIP
    - Requires fixes in multi-server support
- **HELP NEEDED/WANTED**

Further extensions: test with real Resource Manager





Questions