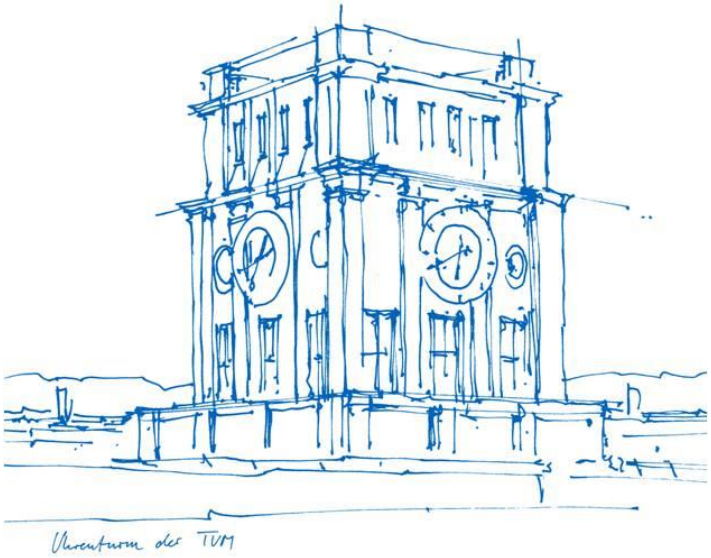
# API Extension and Resource Manager Integration for Malleable MPI Applications




Isaías Alberto Comprés Ureña

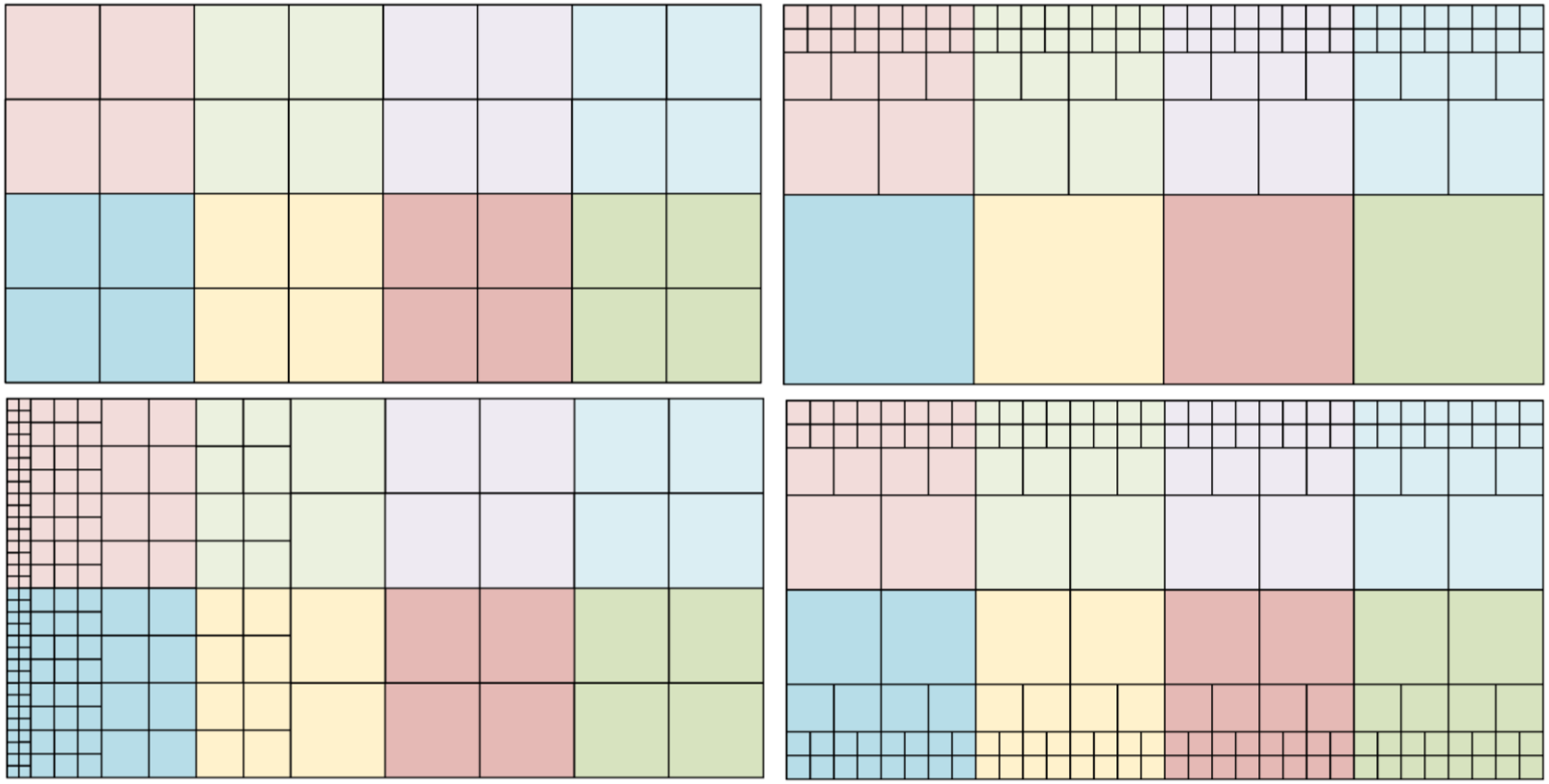
Technical University of Munich

Workbench on Sustained Simulation Performance (WSSP)

2017, Stuttgart

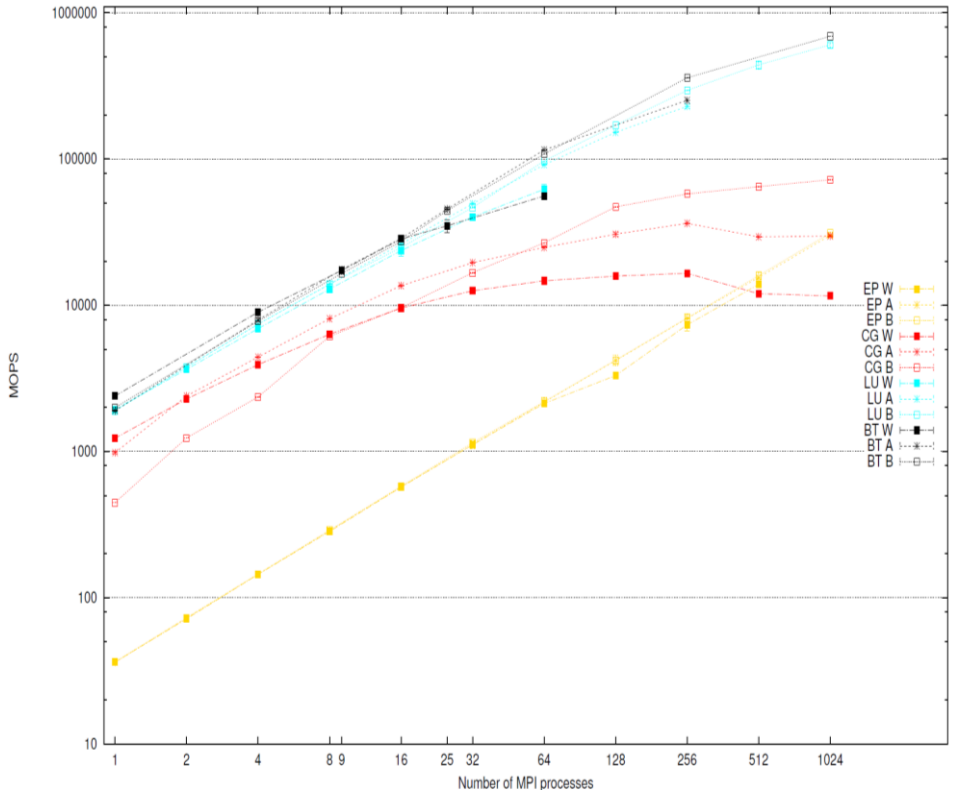
# Motivation (1/3)

Adaptive Mesh Refinement (AMR) applications and variable available parallelism

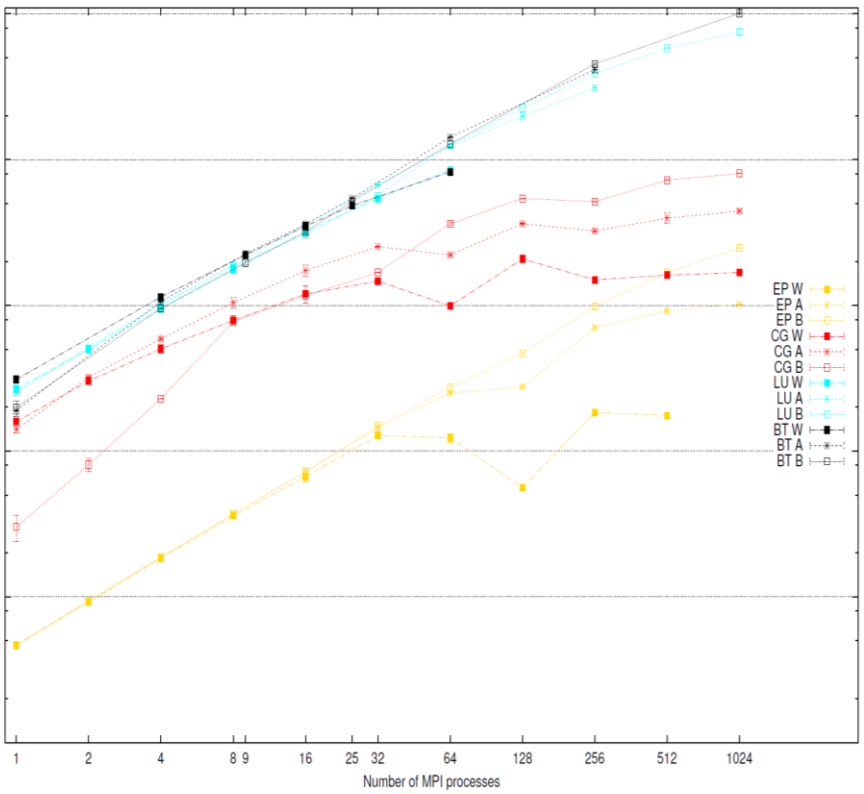


# Motivation (2/3)

Strong scaling applications with one or multiple phases

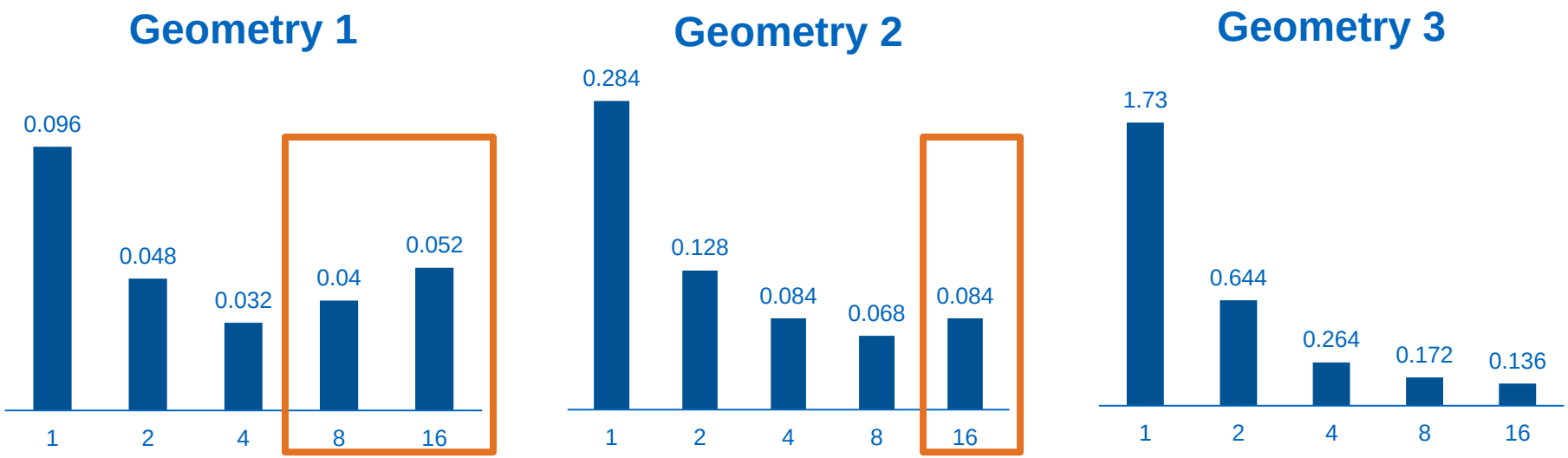



Sandy Bridge




Haswell

# Scaling Example




Scaling depends on input files:

- Size of the domain
- Geometry
- **Unknown at job startup**

Increasing the process count can reduce performance.

# Motivation (3/3)

Efficiency metrics in HPC systems

- Suboptimal network performance due to fixed initial allocations
  - In many network topologies the selection of nodes can impact MPI performance
  - Starting applications early is desirable; idle nodes are undesirable
  - This leads to applications started on suboptimal topologies
  - Resource adaptations of N to N nodes can optimize network performance
- Idle resources due to inflexible resource requirements in jobs
  - Fixed resource counts limit how much resource managers can minimize idle node counts
  - Resource elastic jobs can be used to fill idle nodes
- Energy and power optimizations
  - Power and energy budgets can be met more easily when applications can be scaled based on their energy and power characteristics
  - Energy or power level stabilization techniques can be improved

# Spawn limitations under continuous adaptations

- The spawn operations are synchronous across both the parent and the children process groups.
  - Non-blocking spawn could address this
- These operations produce intercommunicators based on disjoint process groups.
  - Intercomm merge can help, but requires careful comm. management
- Subsequent creations of processes result in additional process groups.
- Destruction of processes can only be done on entire process groups.
- The adaptation of resources can only be initiated by the application.
- Processes created with spawn are typically run in the same resource allocation.
  - This is not an issue with the current API and instead an implementation quality issue.

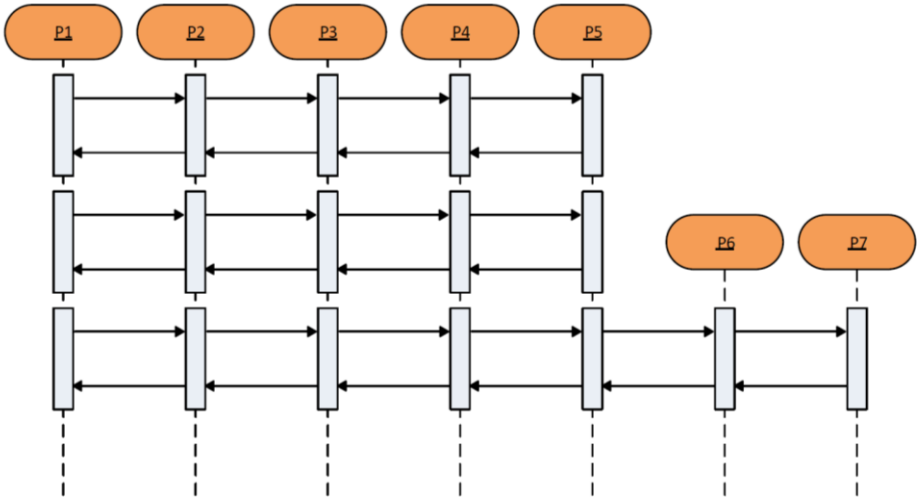
# Proposed Operations

Inversion of control:

- Resource manager initiates and specifies adaptations
- Applications adapt to resource changes only when possible

Only four new operations proposed:

- Initialization in adaptive mode
- Probe for resource adaptation instructions
- Creation of adaptation windows
  - Begin adaptation
  - Commit adaptation



# MPI Extensions Overview

**`MPI_Init_adapt`**(…)

- Initializes the library in adaptive mode

**`MPI_Probe_adapt`**(…)

- Probes the resource manager for adaptations

**`MPI_Comm_adapt_begin`**(…)

- Marks the beginning of an adaptation window
- Provides a set of helper communicators

**`MPI_Comm_adapt_commit`**(…)

- Marks the end of an adaptation window
- Sets adapted `MPI_COMM_WORLD`

**Code Structure**

```
MPI_Init_adapt(…, &status);
for (…) {
    MPI_Probe_adapt(&adapt,…);
    if(adapt) {
        MPI_Comm_adapt_begin(…);
        // redistribution code
        MPI_Comm_adapt_commit(…);
    }
    // compute and MPI code
}
```

# Initialization in Adaptive Mode

```
int MPI_Init_adapt (
        int * argc ,
        char *** argv ,
        int * status
);
```

*status*:

- New
- Joining

**Code Structure**

```
MPI_Init_adapt(…, &status);
for (…){
    MPI_Probe_adapt(&adapt,…);
    if(adapt){
        MPI_Comm_adapt_begin(…);
        // redistribution code
        MPI_Comm_adapt_commit(…);
    }
    // compute and MPI code
}
```

# Probing for Adaptation Data

```
int MPI_Probe_adapt (
        int * operation,
        int * status,
        MPI_Info * info
);
```

*operation*:

- Expansion
- Reduction
- Combined
- Migration

*status*:

- New
- Joining
- Staying
- Leaving

**Code Structure**

```
MPI_Init_adapt(…, &status);
for (…){
    MPI_Probe_adapt(&adapt,…);
    if(adapt){
        MPI_Comm_adapt_begin(…);
        // redistribution code
        MPI_Comm_adapt_commit(…);
    }
    // compute and MPI code
}
```

# Beginning an Adaptation

```
int MPI_Comm_adapt_begin (
        int * intercomm,
        int * future_comm_world
);
```

*intercomm*:

- Equivalent to MPI_Comm_spawn
- Used to reach parents from the child group, and the children from the parent group

**Code Structure**

```
MPI_Init_adapt(…, &status);
for (…){
    MPI_Probe_adapt(&adapt,…);
    if(adapt){
        MPI_Comm_adapt_begin(…);
        // redistribution code
        MPI_Comm_adapt_commit(…);
    }
    // compute and MPI code
}
```

*future_comm_world*:

- Contains all staying parents plus children processes
- Parent processes that are leaving receive MPI_COMM_NULL

# Probing for Adaptation Data

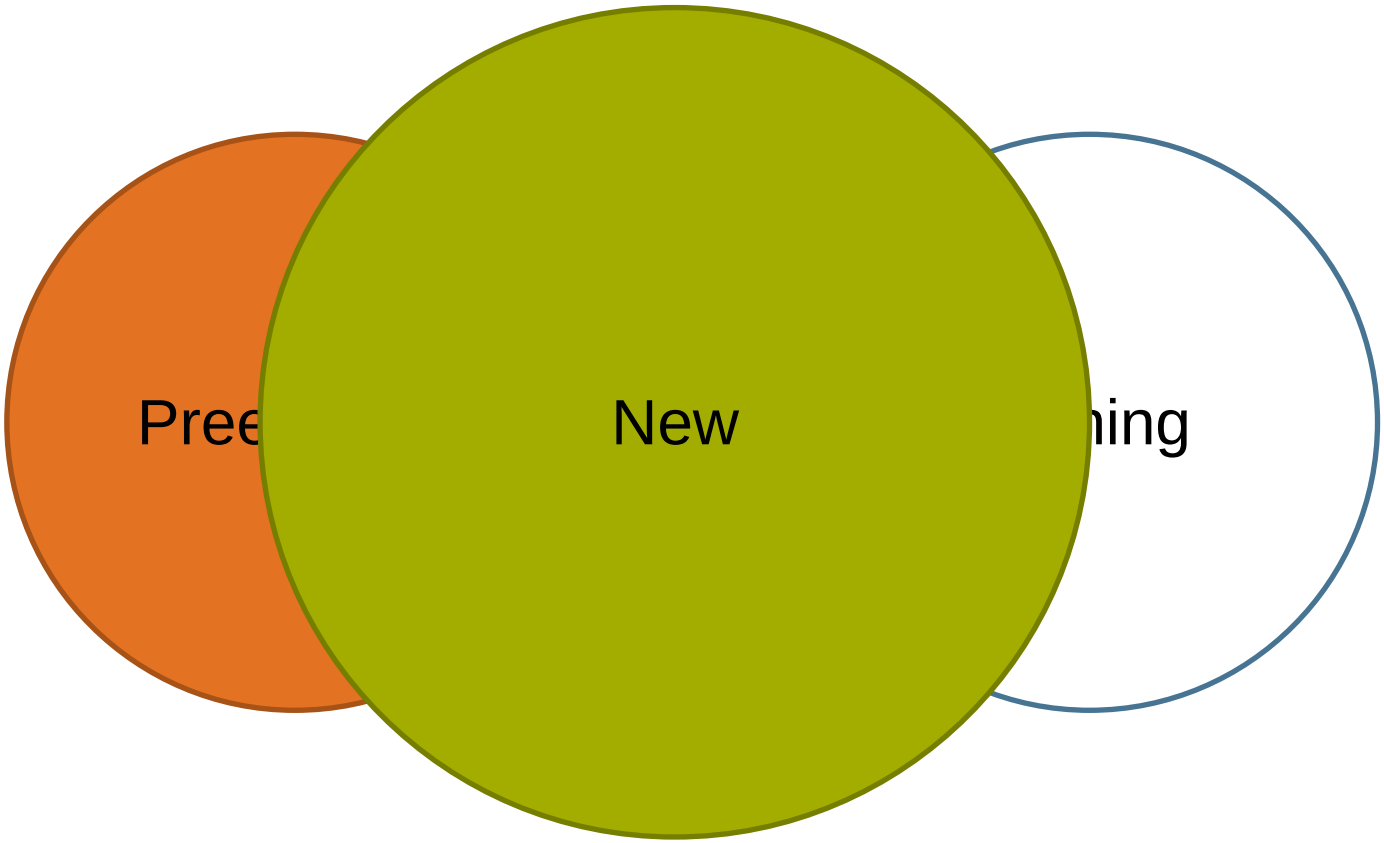
```
int MPI_Comm_adapt_commit ();
```

- MPI_COMM_WORLD is set to the *new_comm_world* communicaton provided by the MPI_Comm_adapt_begin operation earlier.
- Leaving processes are required to terminate
  - In our current prototype the operation itself calls *exit()*
  - Our current applications clean up memory and file descriptors in the adaptation window, before commit

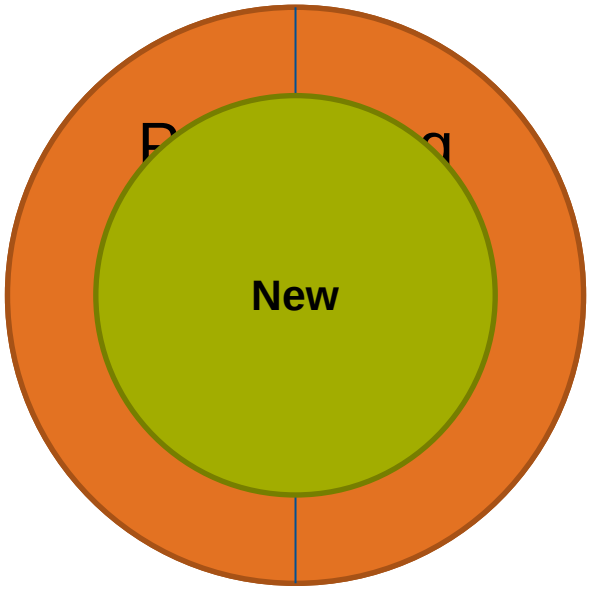
**Code Structure**

```
MPI_Init_adapt(…, &status);
for (…){
    MPI_Probe_adapt(&adapt,…);
    if(adapt){
        MPI_Comm_adapt_begin(…);
        // redistribution code
        MPI_Comm_adapt_commit(…);
    }
    // compute and MPI code
}
```

# Resource Adaptation: Addition



# Resource Adaptation: Subtraction



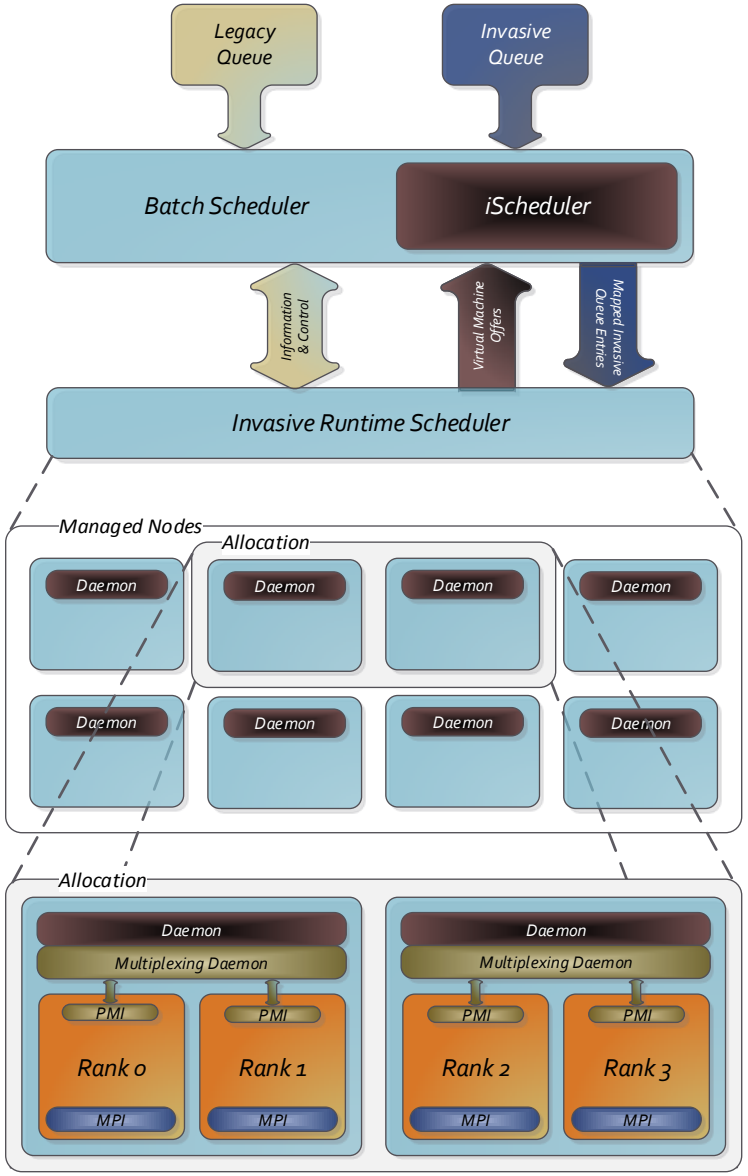
# Resource Management Integration

**iMPI:**

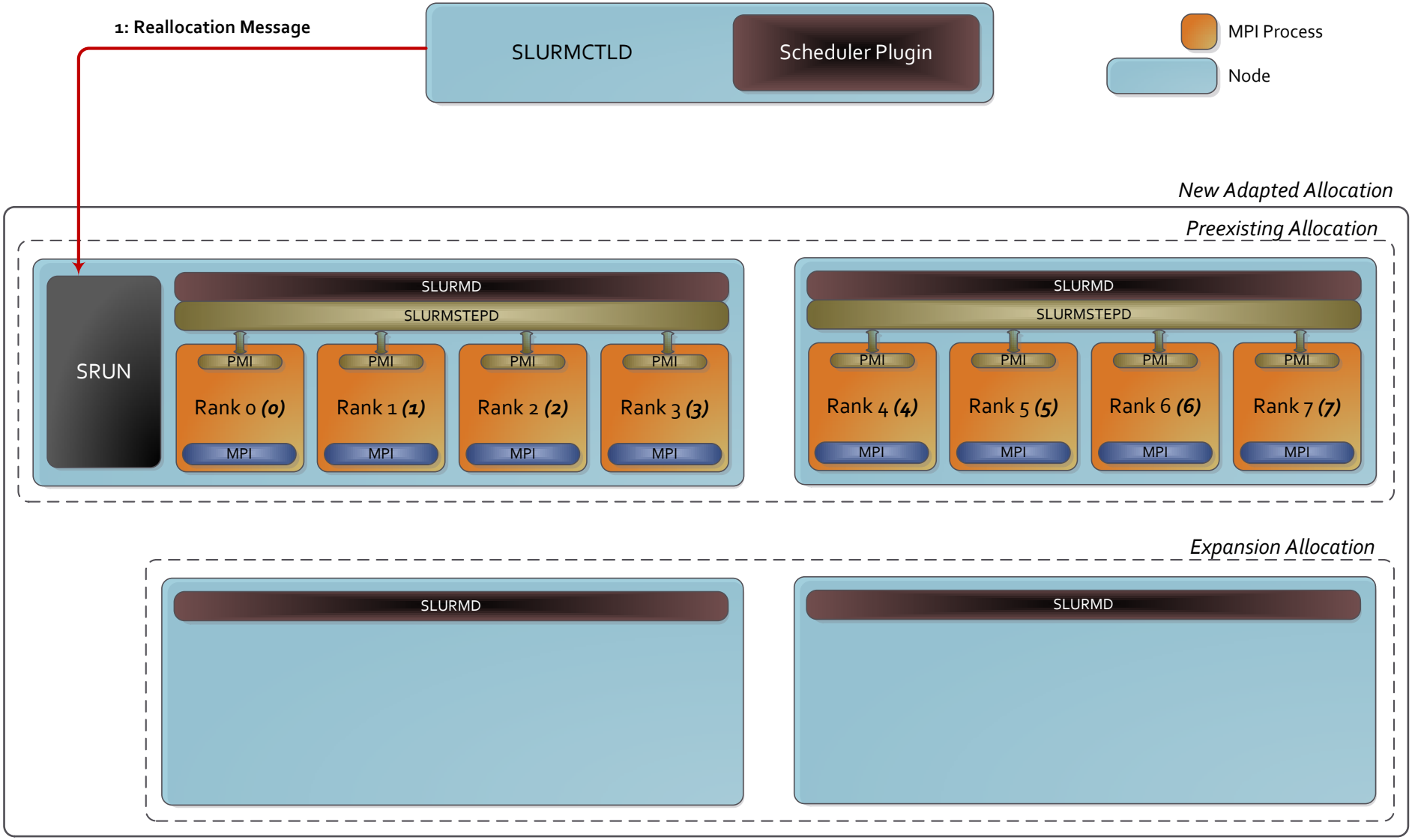
- MPI with new extensions
- Based on MPICH 3.2
  - Includes Fortran wrappers
- PMI2 integration with iRM

**iRM:**

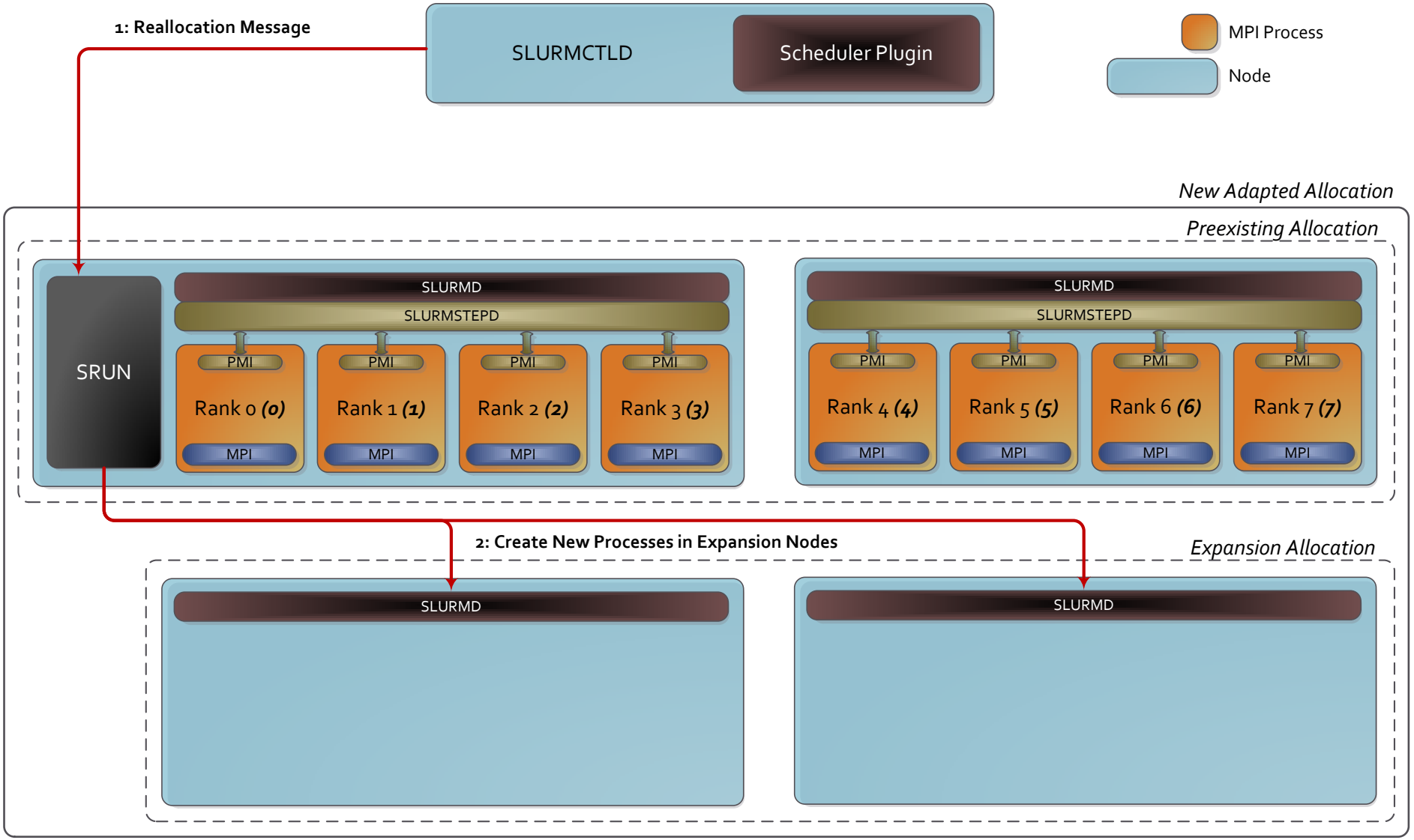
- Based on SLURM 16.08
  - Split batch and runtime schedulers
  - Distributed system with multiple daemons, commands and other binaries
- SuperMUC target
  - Partition emulation under Load Leveler
  - Dynamic host list configuration
  - slurm.conf generation
  - Daemon bootstrap
- Workstation target
  - Virtual machine hosts
  - Local system as login node
- Experimental scheduler and monitor



# Adaptation Step 1

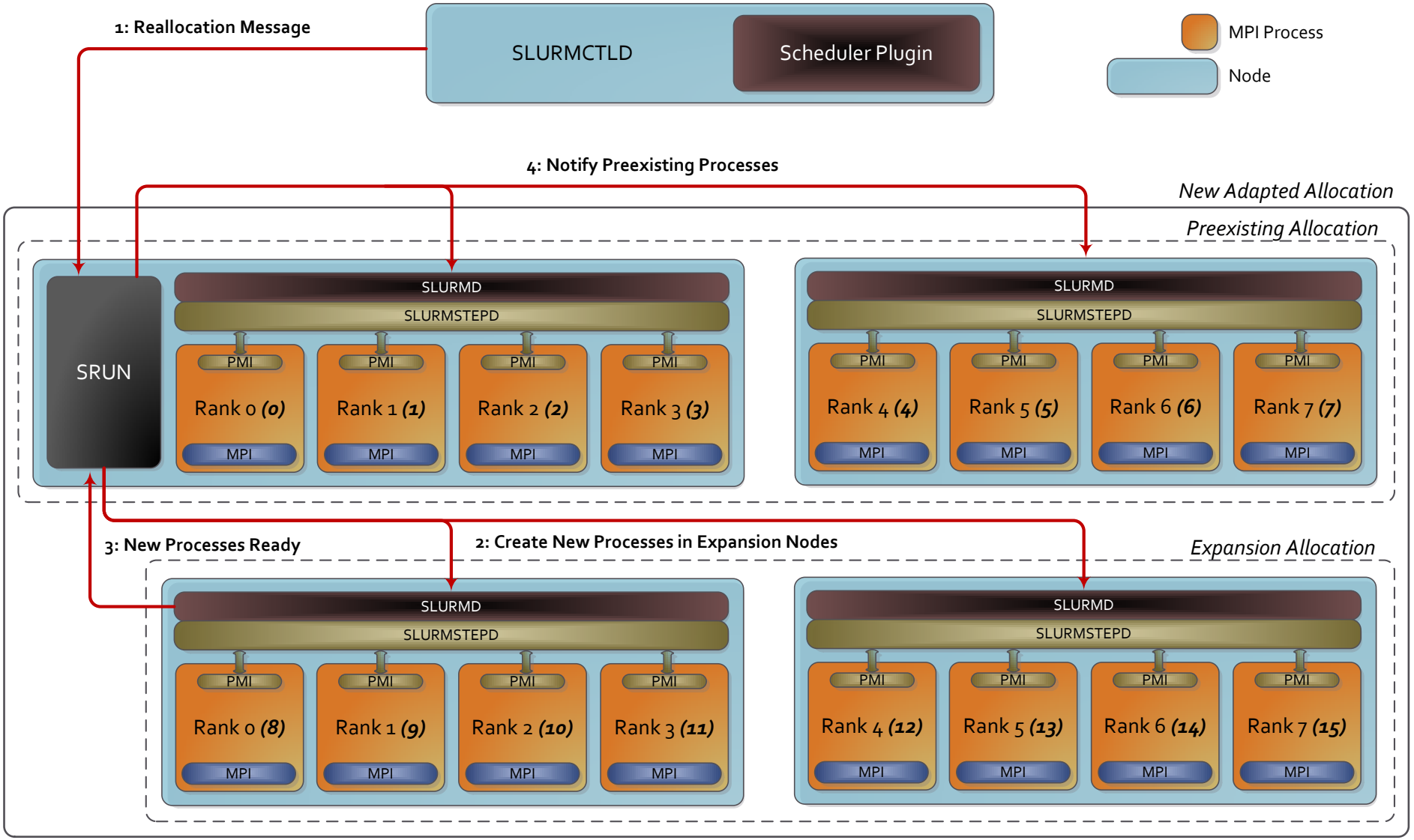


# Adaptation Step 2

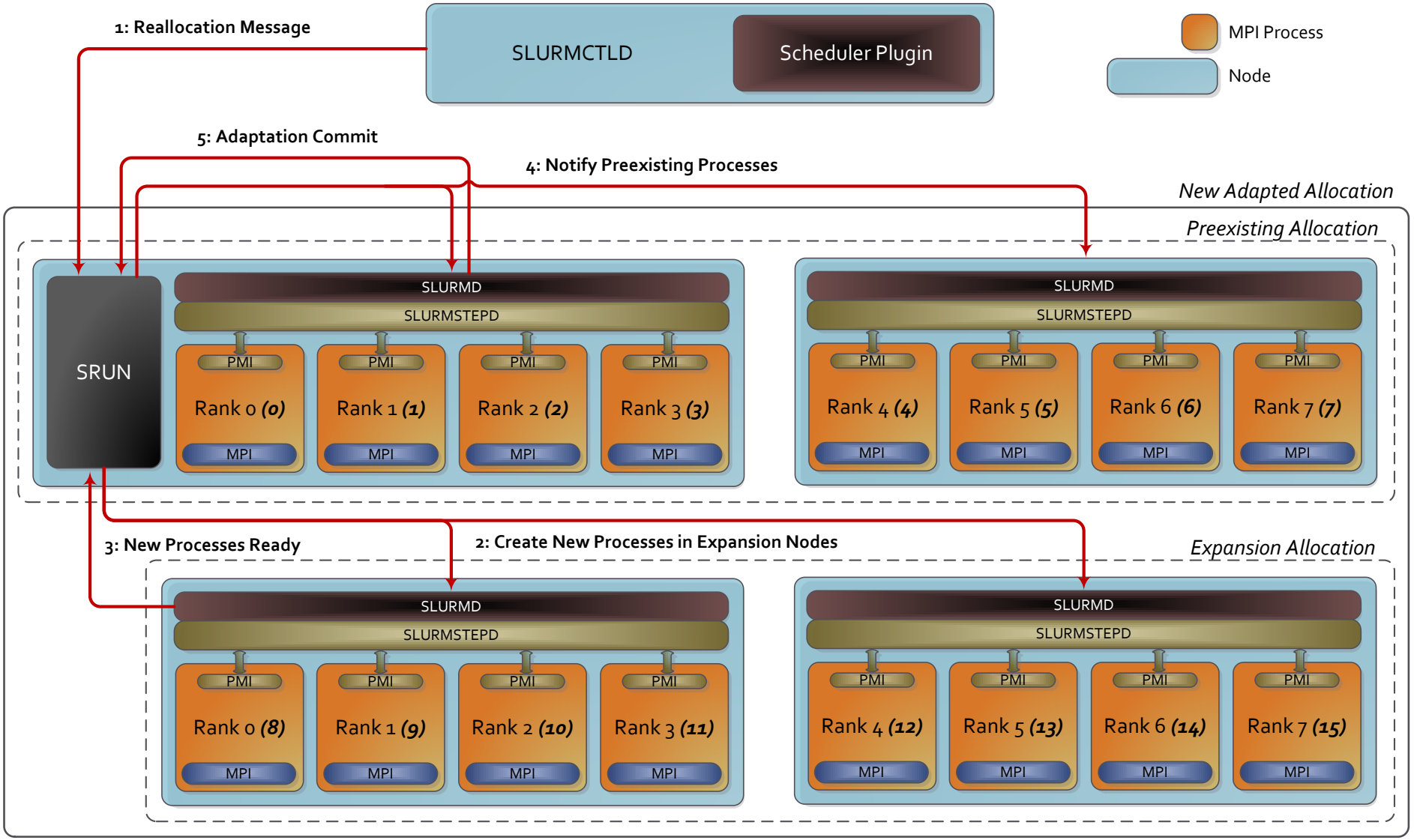


# Adaptation Step 3

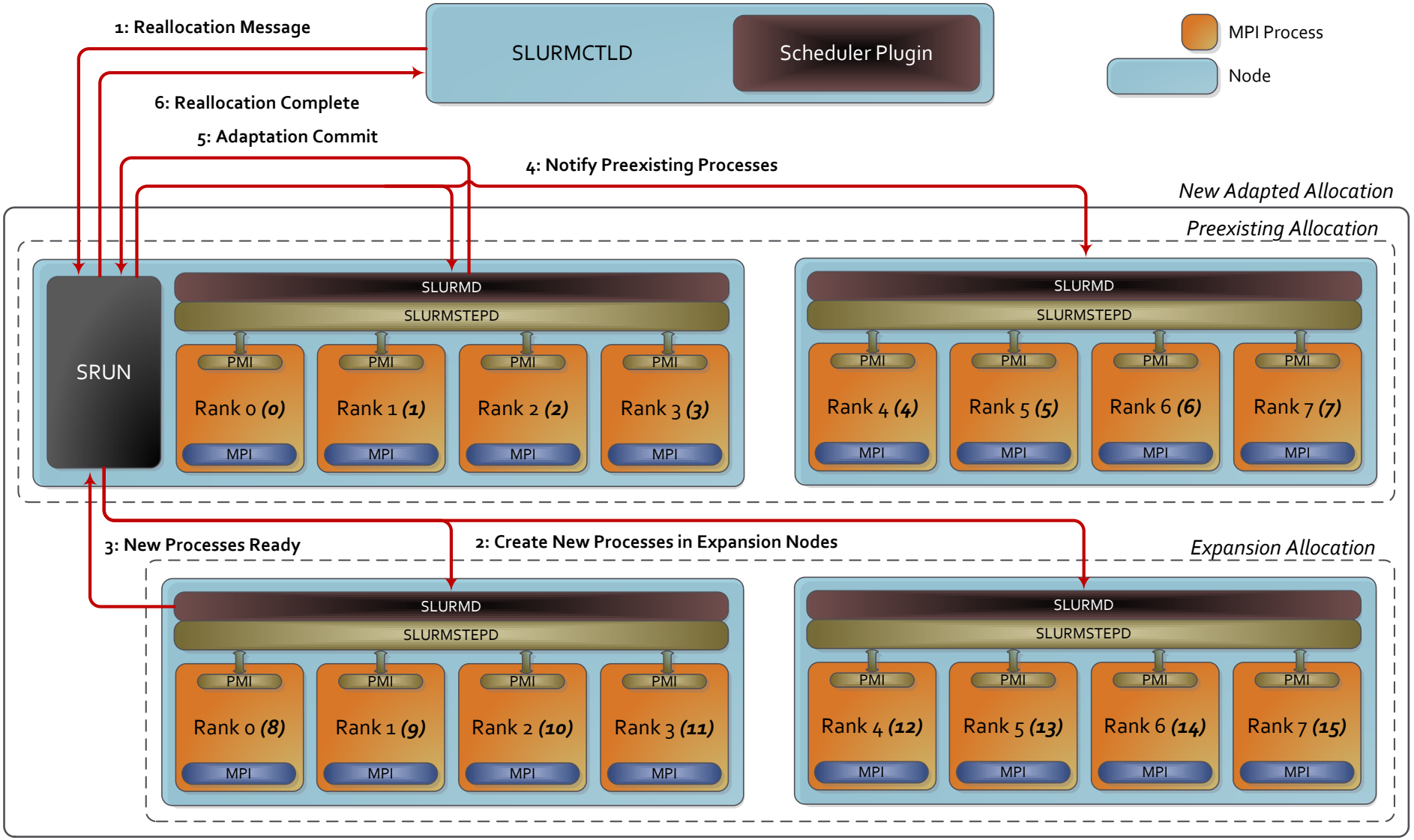
# Adaptation Step 4



# Adaptation Step 5



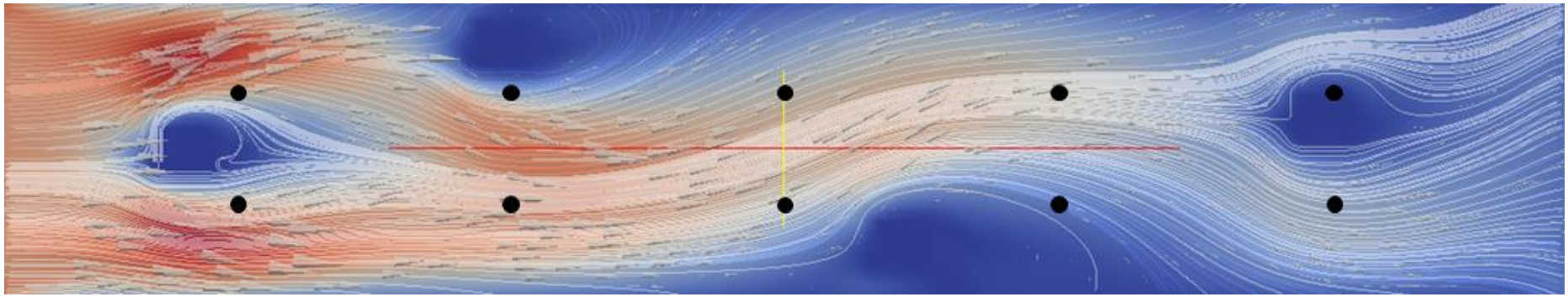
# Adaptation Step 6



# Application Performance:
## Simulation and Inverse Problem Solver

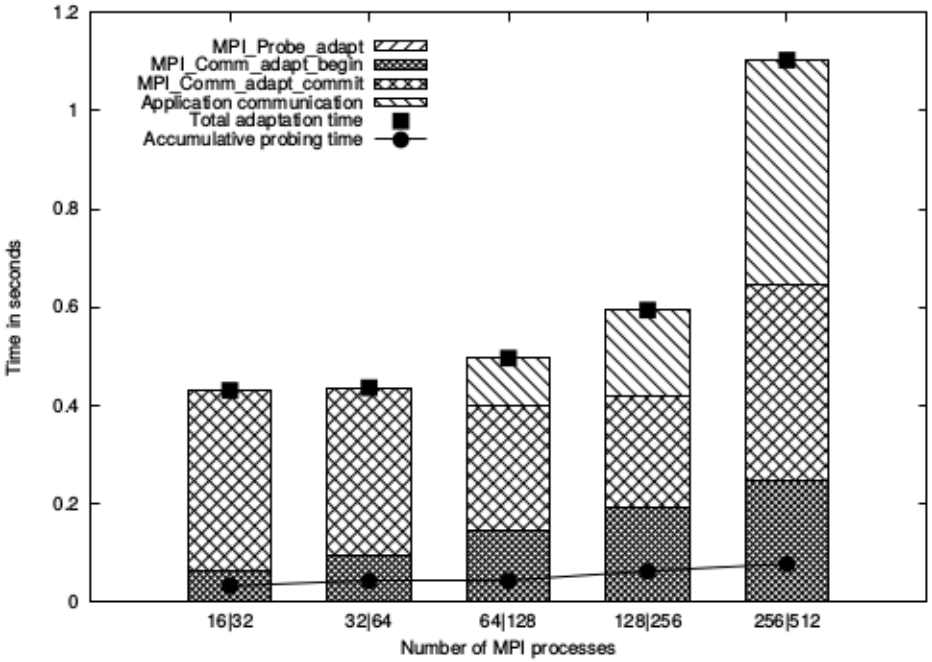
Elastic Surrogate Model Construction for a Statistical Inverse Problem

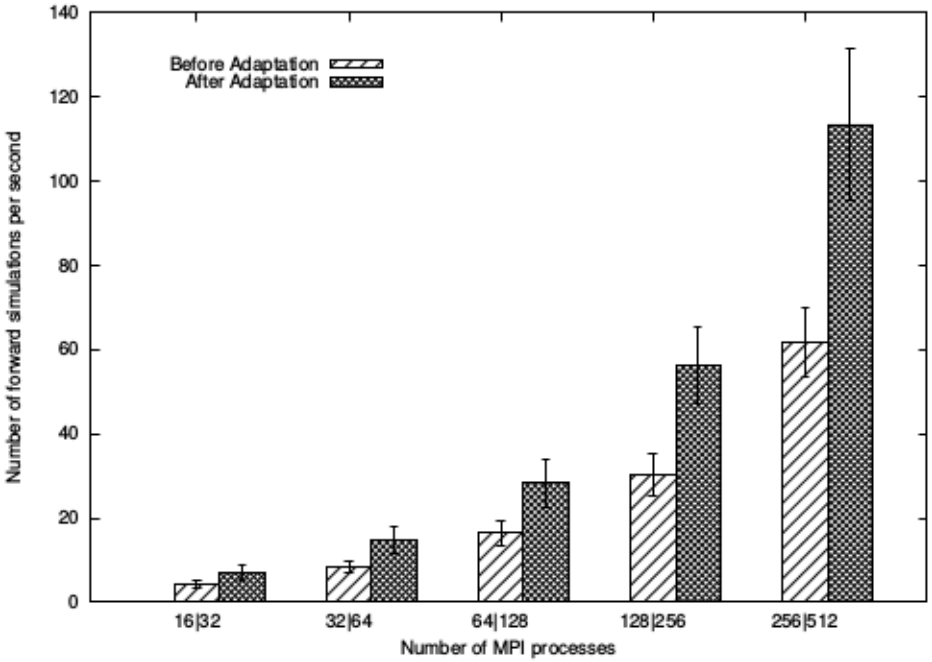
- Locates the number of obstacles in a fluid channel
- 2D version as first elastic conversion
- Fluid simulation as input, instead of real physical flow
  - Quick setup of various experimental scenarios
  - Easy verification of the method‘s success
- Outputs the predicted obstacles location



Mo-Hellenbrand, Bungartz

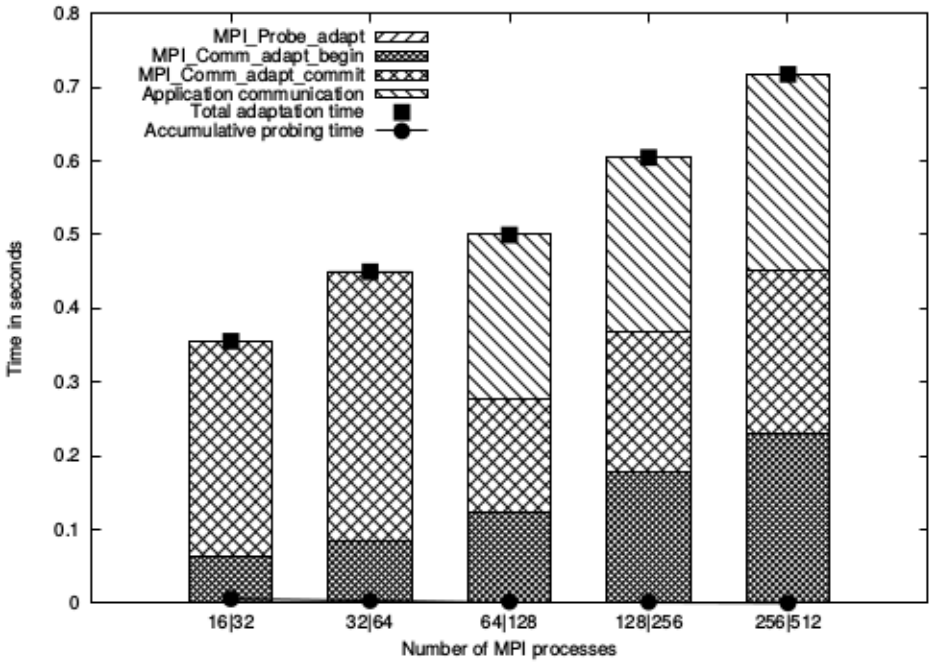
# Application Performance:
## Simulation and Inverse Problem Solver

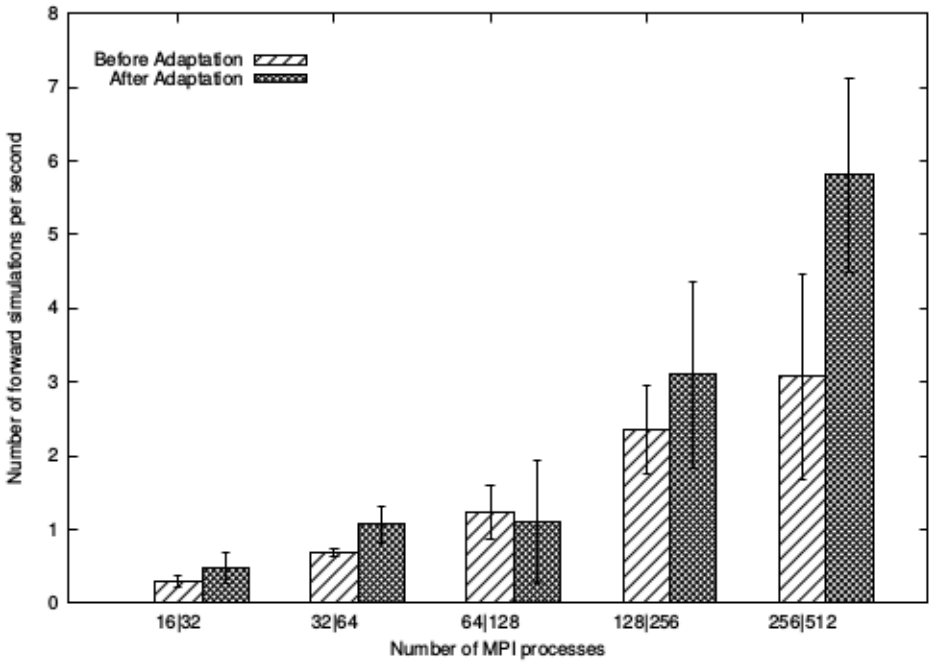



Mo-Hellenbrand, Bungartz

# Application Performance:
## Simulation and Inverse Problem Solver

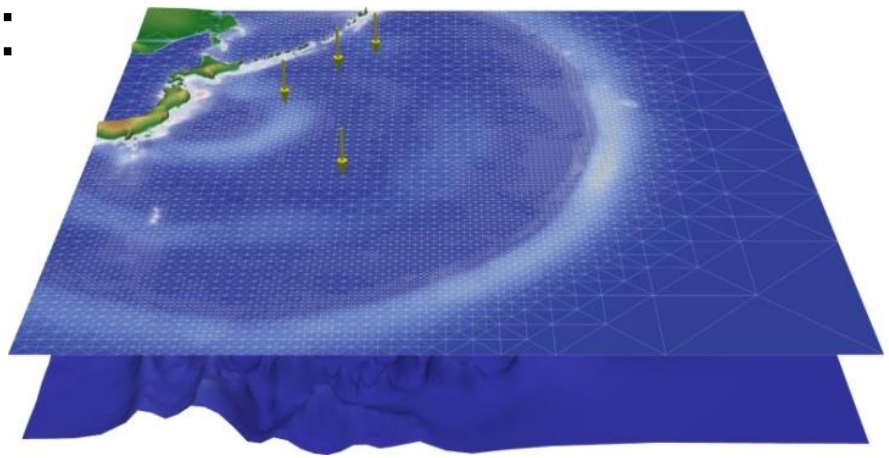


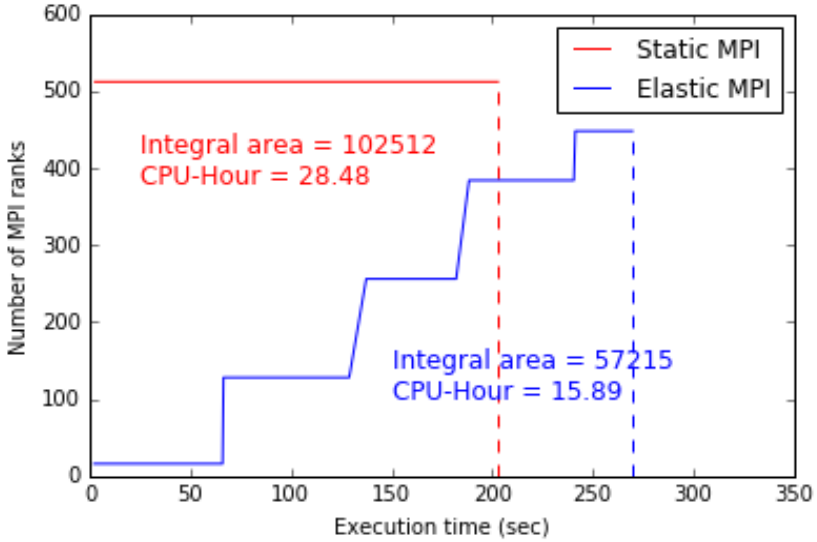



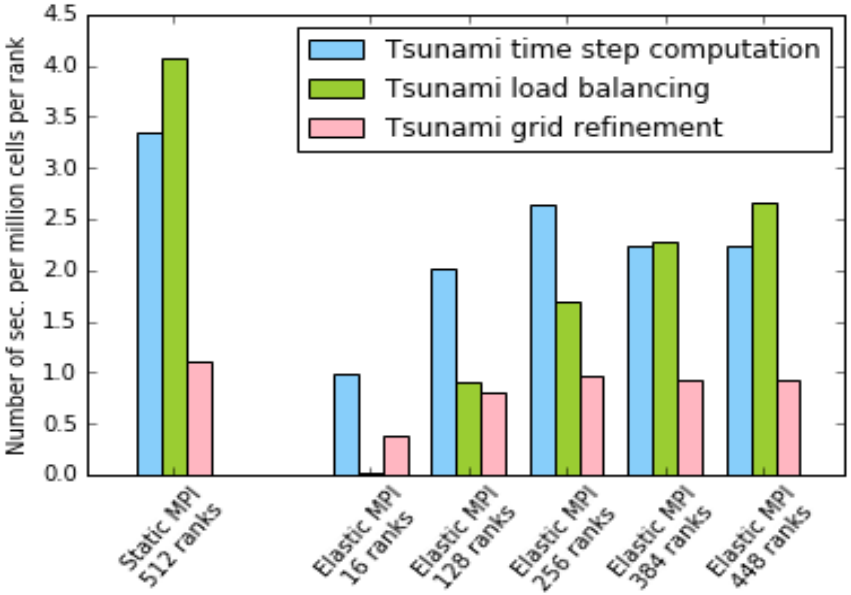

Mo-Hellenbrand, Bungartz

# Application Performance:

## Tsunami Simulation

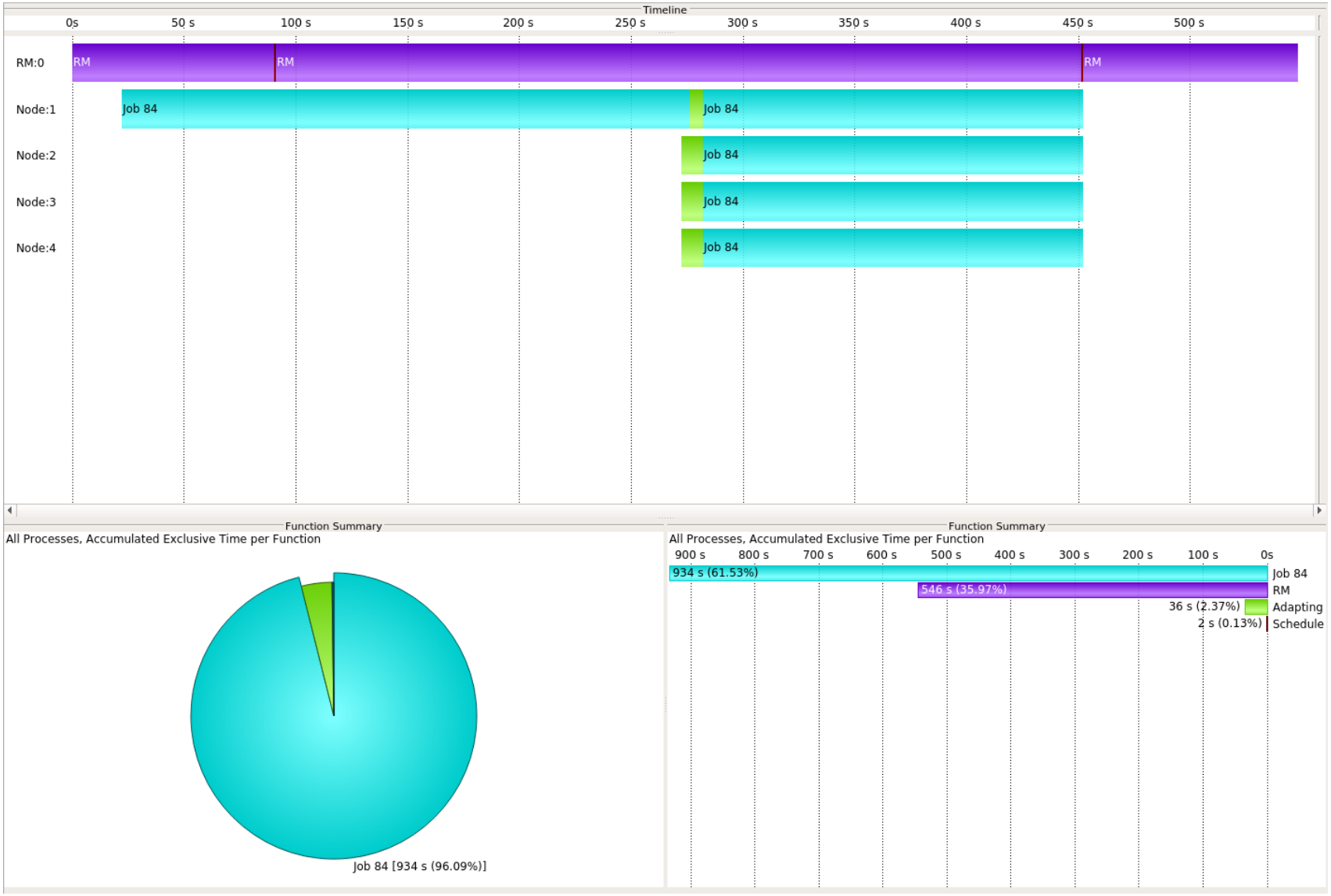




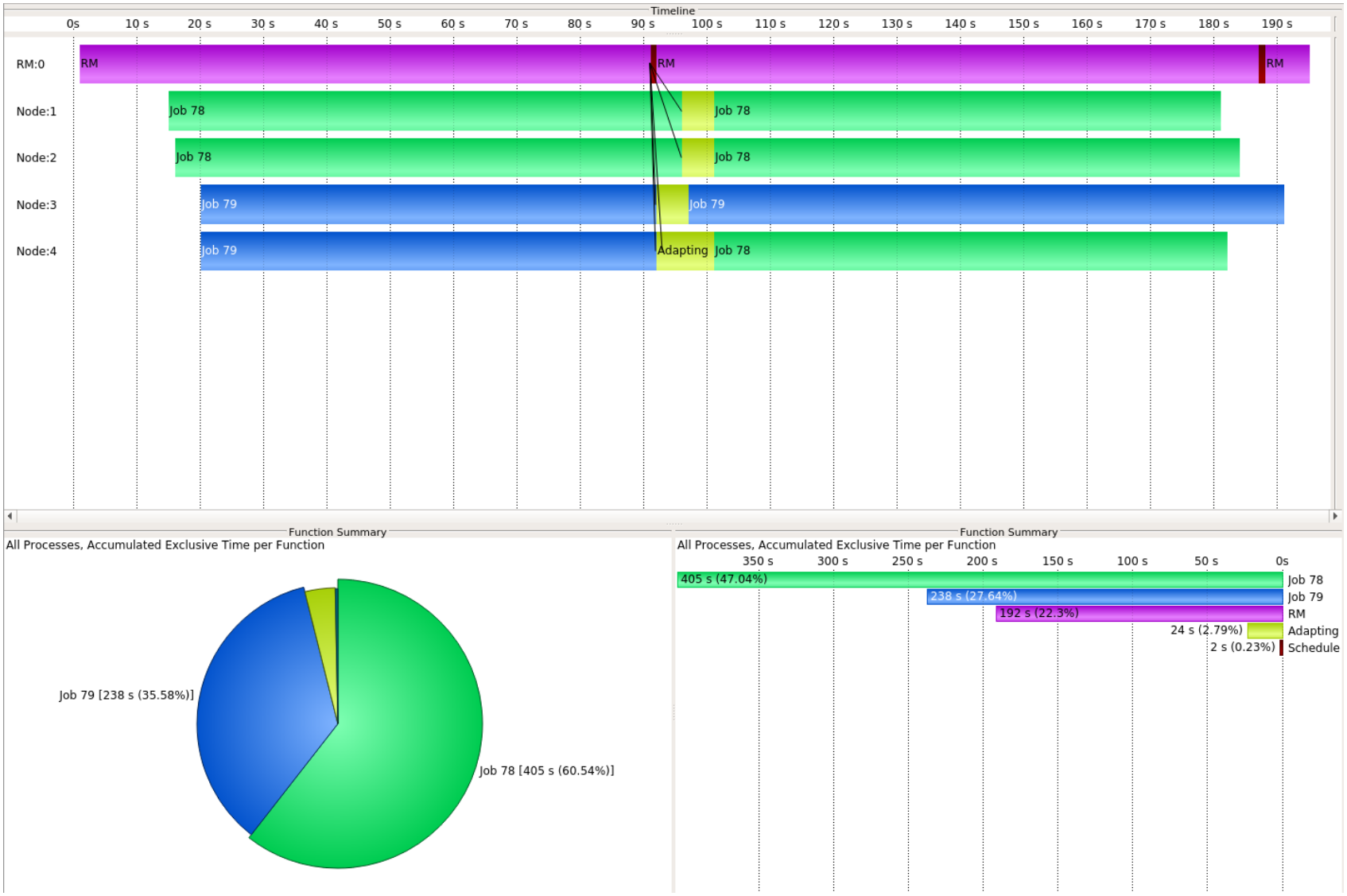




Mo-Hellenbrand, Bungartz

# Adaptation Visualization



# Adaptation Visualization



# Conclusion

Resource-Elastic support requires vertical integration:

- Application development
- Programming model
- Runtime system
- Resource management infrastructure
- Scheduling
- Libraries

Can help improve sustained performance in the future:

- System wide performance
  - Minimization of idle node counts
  - Energy-aware scheduling
- Application specific performance
  - Allocations for acceptable parallel efficiency

# Questions and Answers

Thank you for your attention.