D3

Infrastructure and API Extensions for Elastic Execution of MPI Applications

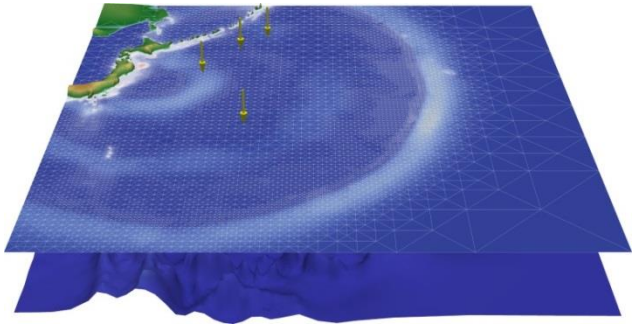
Isaías Comprés, Ao Mo-Hellenbrand,
Michael Gerndt, Hans-Joachim Bungartz

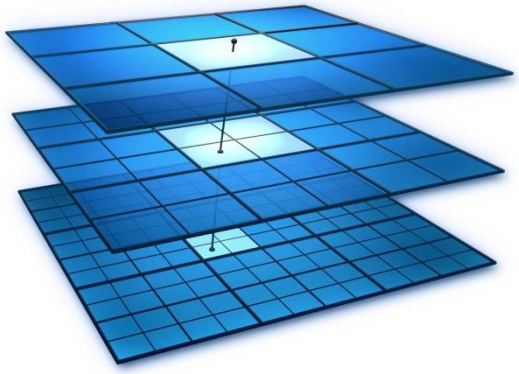
EuroMPI 2016, Edinburgh, Scotland

InvasIC: D3 Subproject Overview

D3

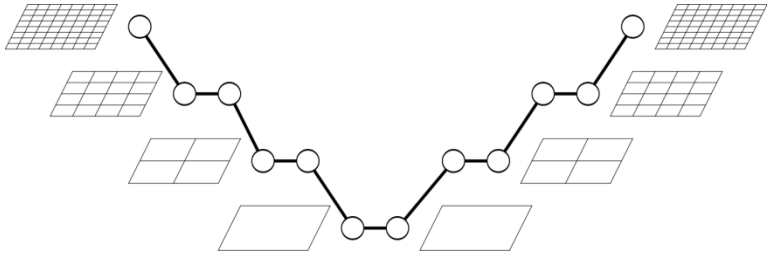
Invasive HPC applications

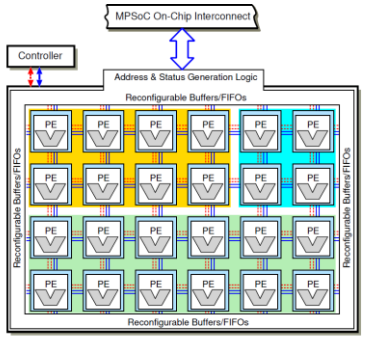




Invasive resource management for HPC

Invasive X10 applications on MPSoCs





Resource Adaptivity for MPI Applications

Motivation

- State of the art scientific applications that utilize algorithms with evolving scalability properties
- The current assignment of fixed resources to these applications is suboptimal

Implementation

- **MPI extensions**
  - Extend the API with adaptive operations
- **MPI Library**
  - Based on MPICH 3.2
- **Resource Manager**
  - Based on SLURM 15.08

MPI Extensions Overview

D3

Proposed 4 new operations as an extension to the MPI standard:

`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

D3

```
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

D3

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

operation:

- Addition
- Reduction
- Combined
- Migration

status:

- 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
}
```

Probing for Adaptation Data with FT

D3

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

operation:

- Expansion
- Reduction
- Combined
- Migration
- ***Fault***

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
}
```

Probing for Adaptation Data

D3

```
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

D3

```
int MPI_Comm_adapt_commit ();
```

- MPI_COMM_WORLD is set to the *new_comm_world* communicator provided by the **MPI_Comm_adapt_begin** operation earlier.

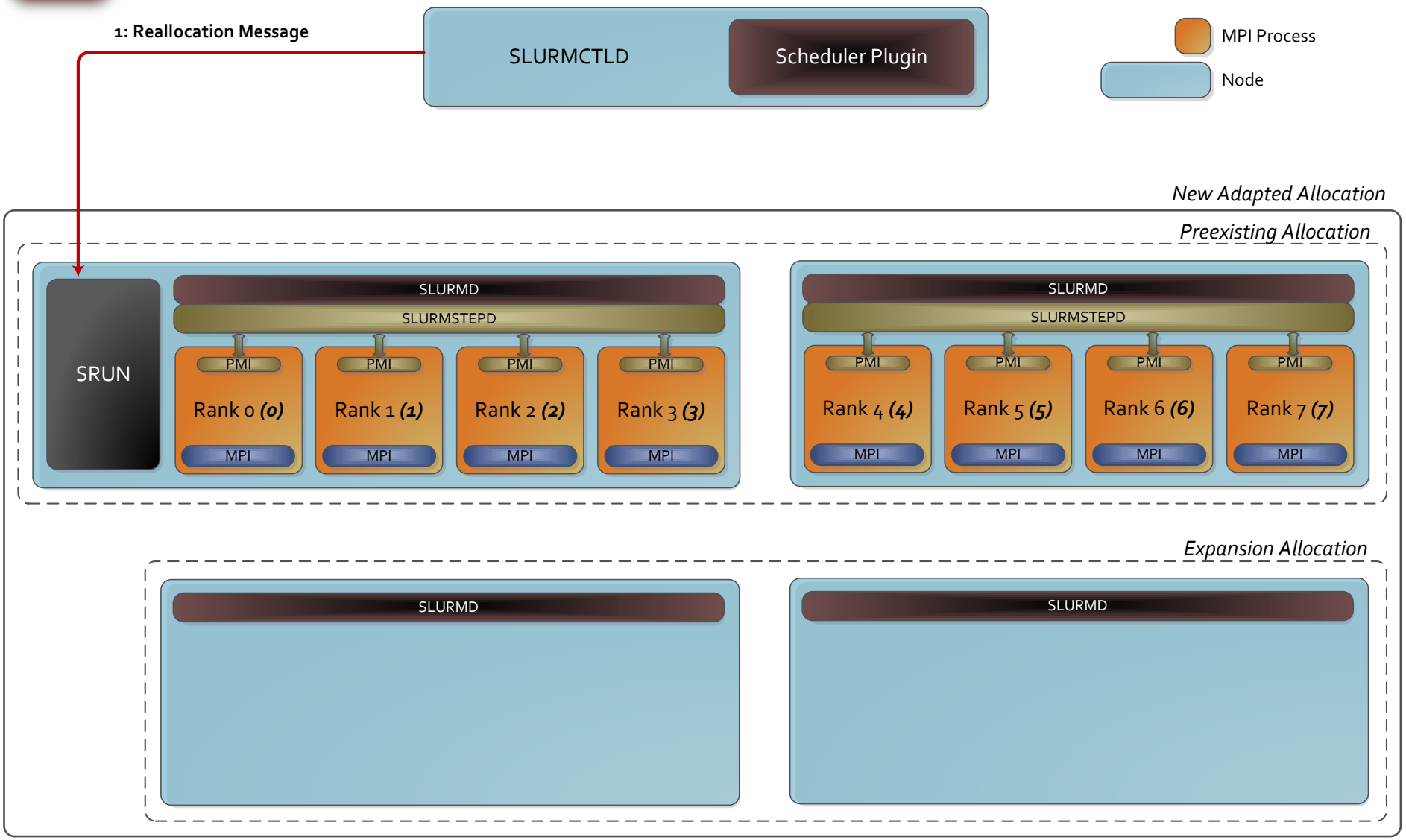
- Leaving processes are required to terminate
  - In our current prototype the operation itself calls *exit()*
  - Our current applications clean up memory and file descriptors in the adaptation window, before commit

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
}
```

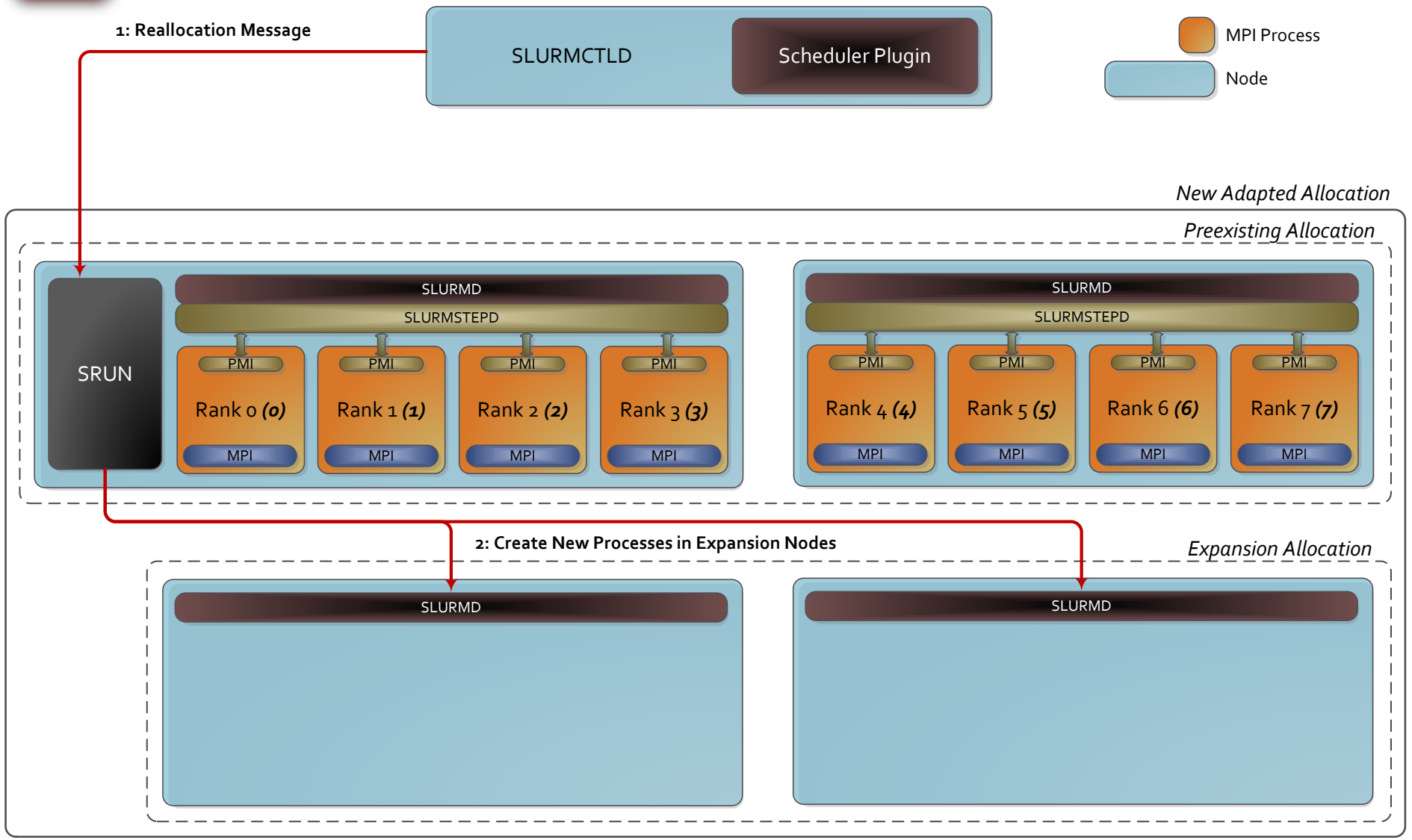
Adaptation Step 1

D3



Adaptation Step 2

D3

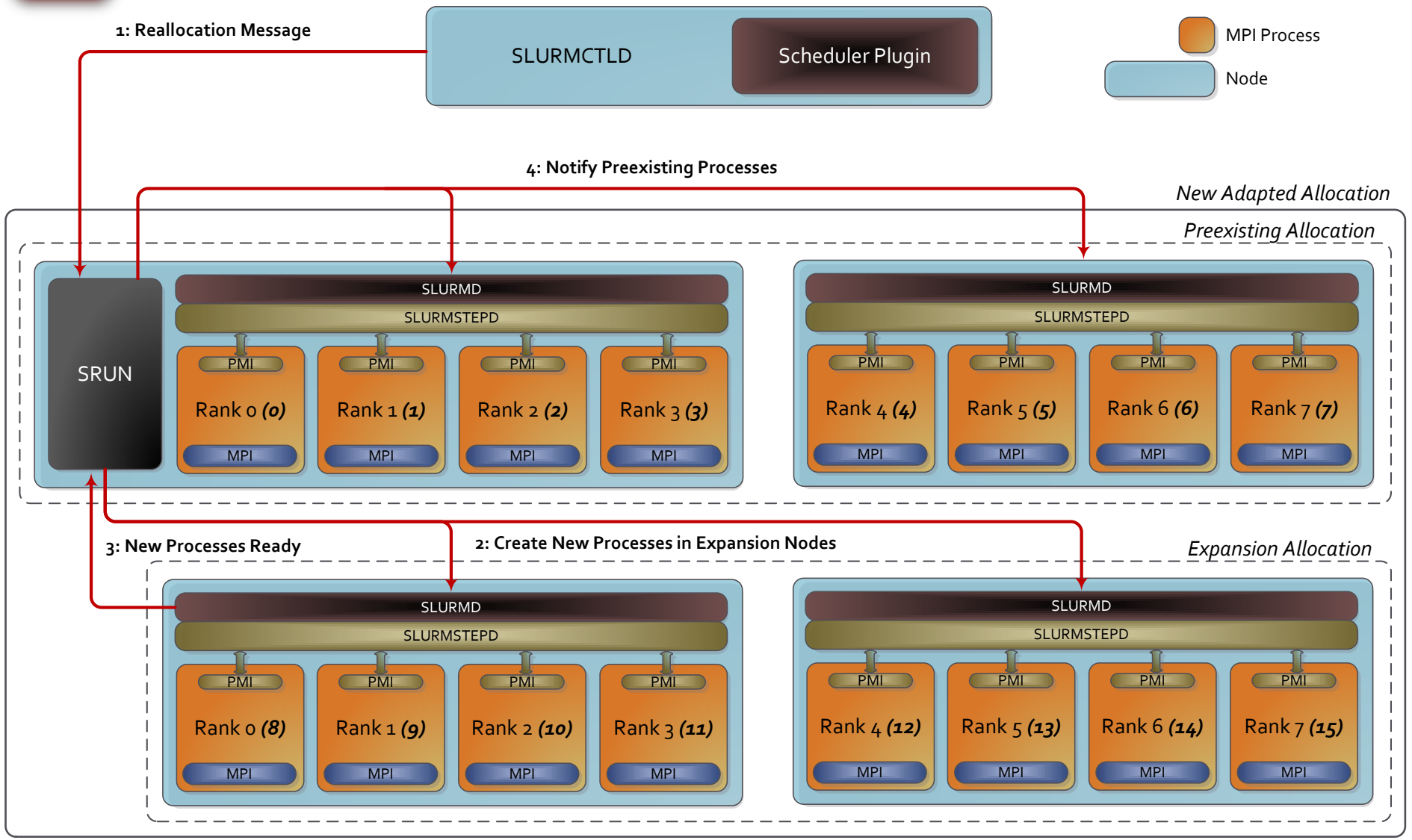


Adaptation Step 3

D3

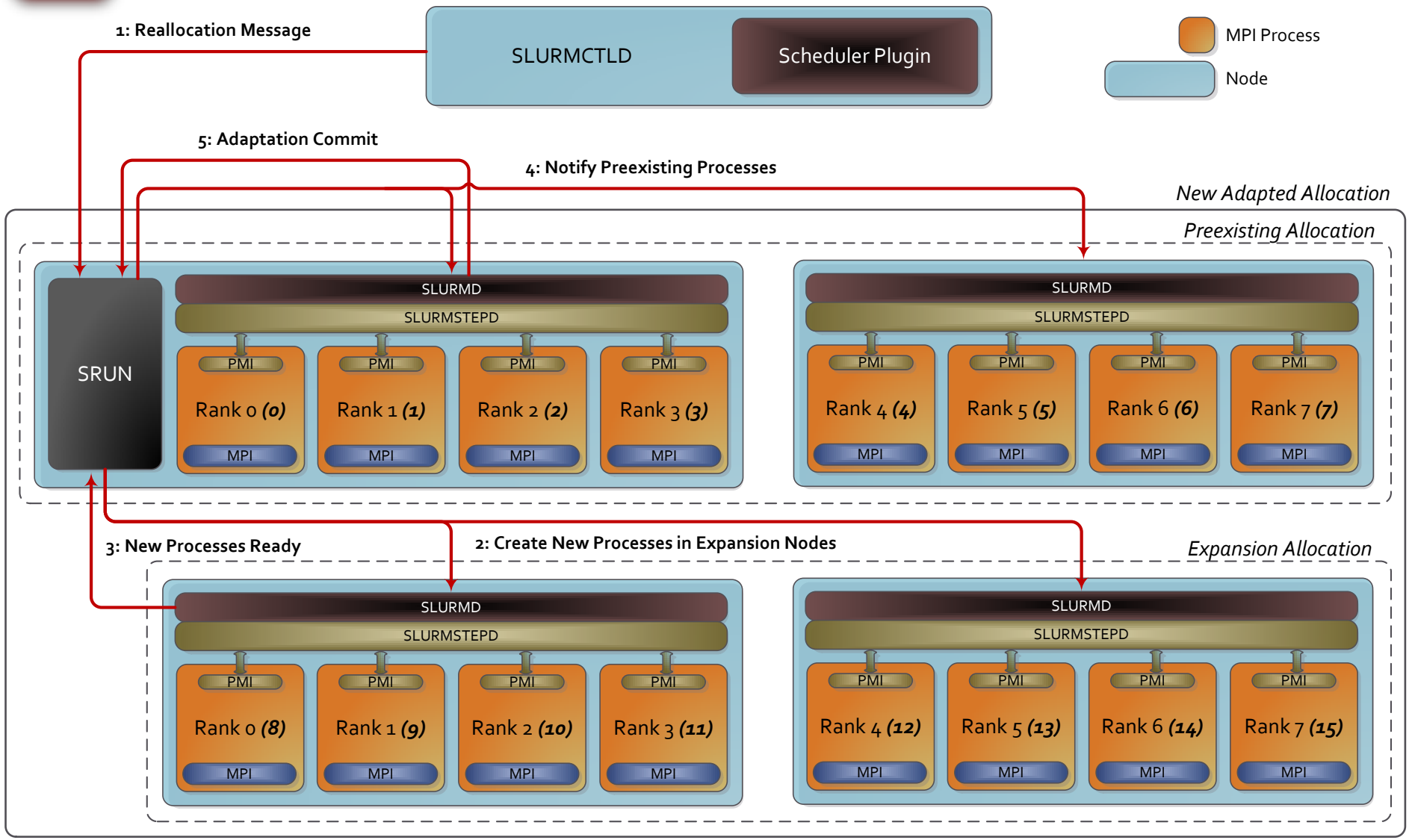
Adaptation Step 4

D3



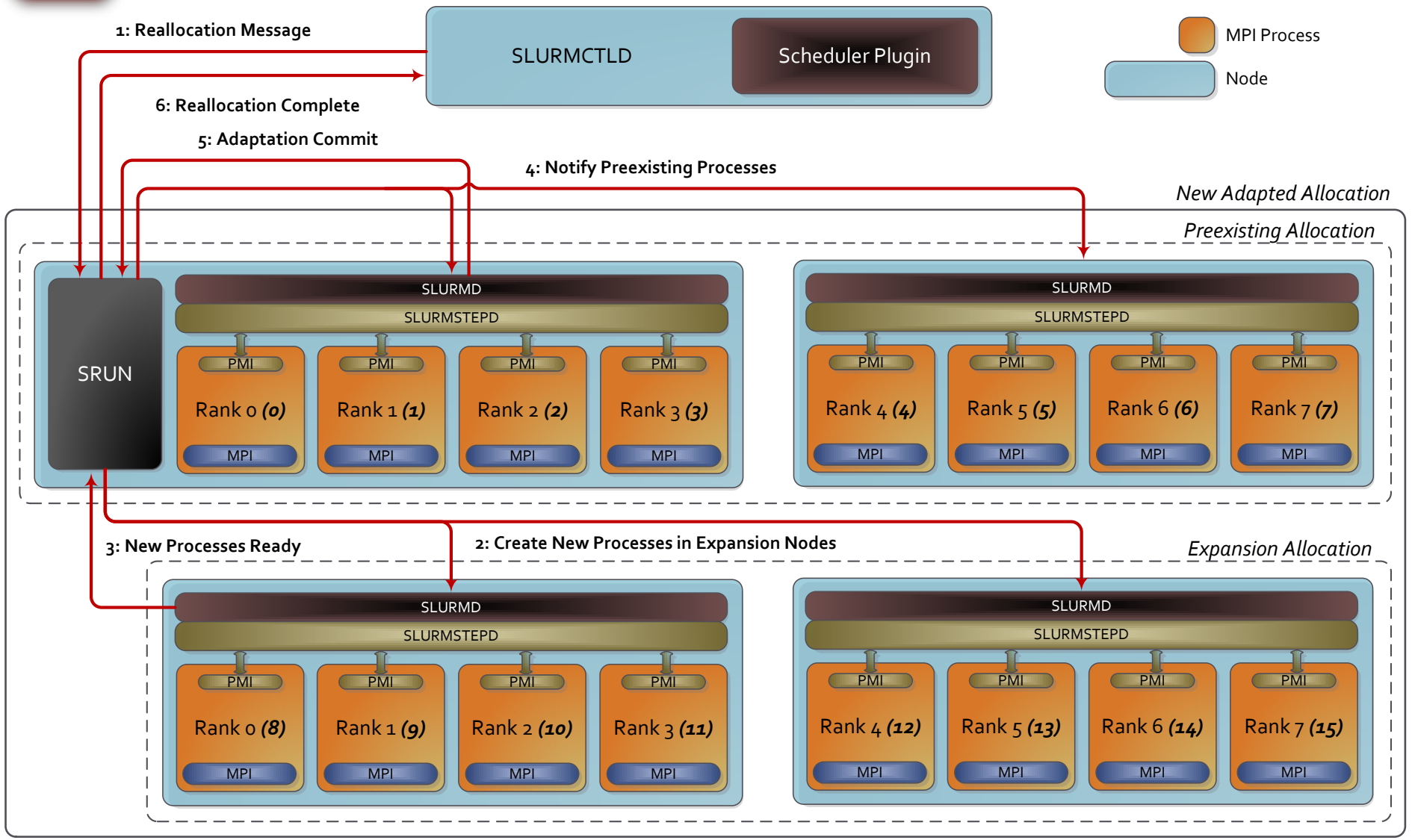
Adaptation Step 5

D3



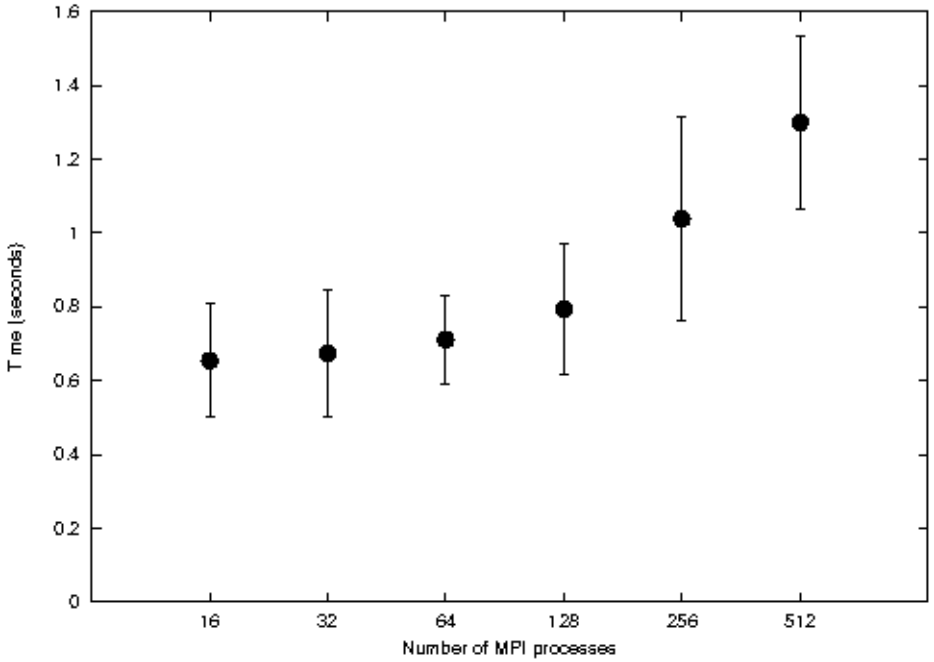
Adaptation Step 6

D3

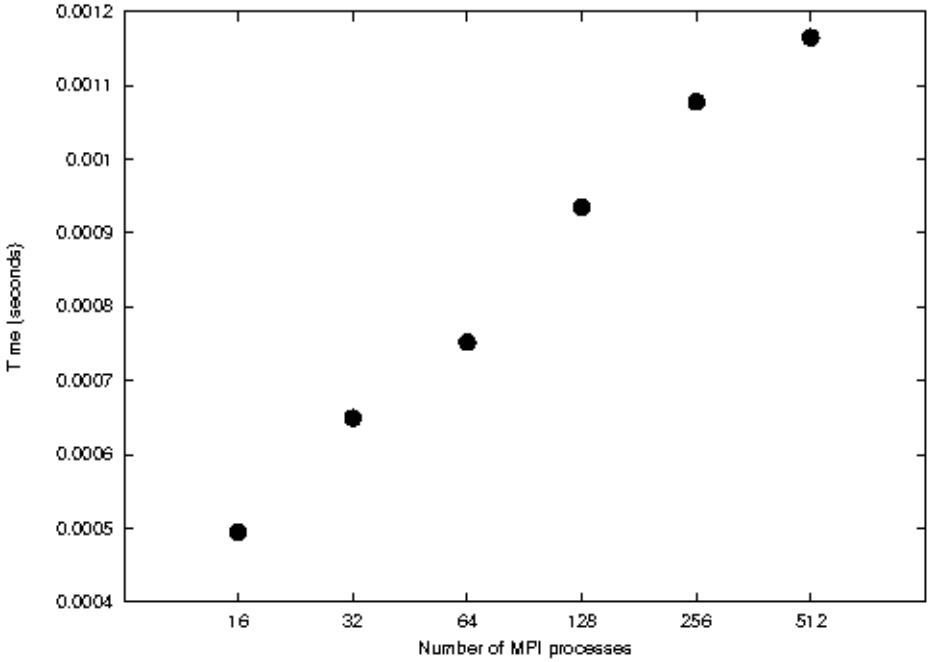


Performance Results

D3



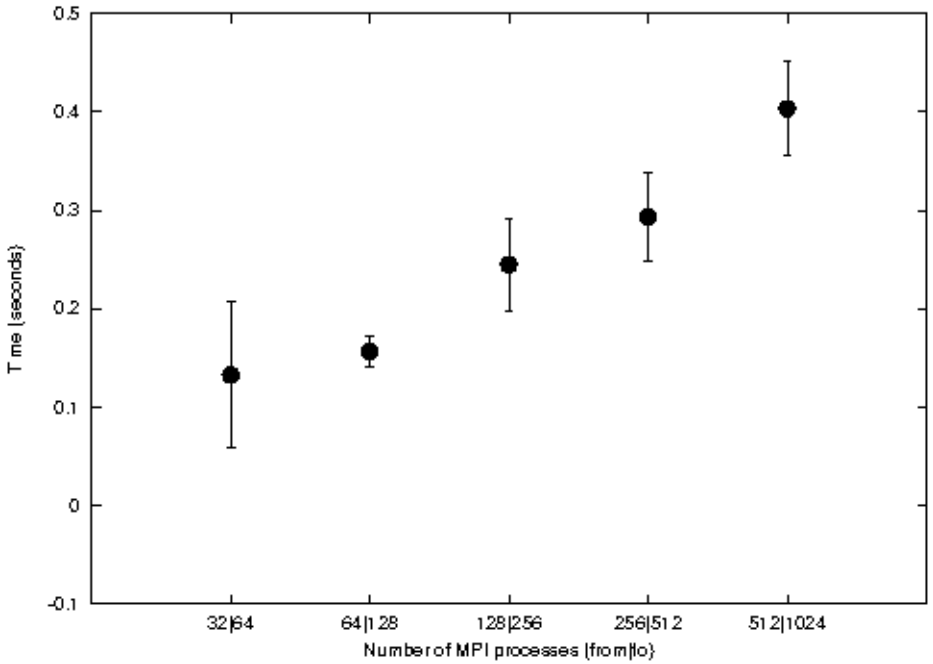
MPI_Init_adapt



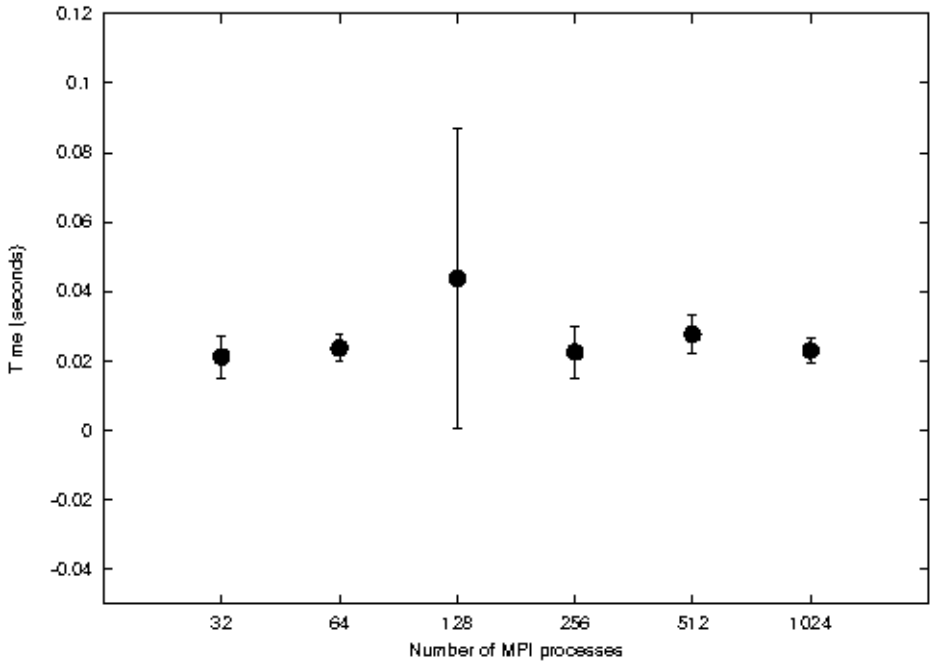
MPI_Probe_adapt

Performance Results

D3



MPI_Comm_adapt_begin

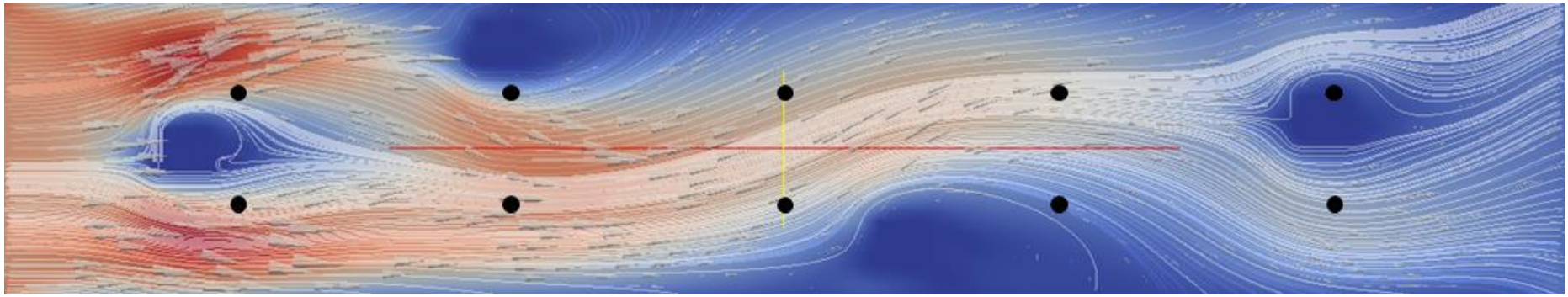


MPI_Comm_adapt_commit

Performance Results

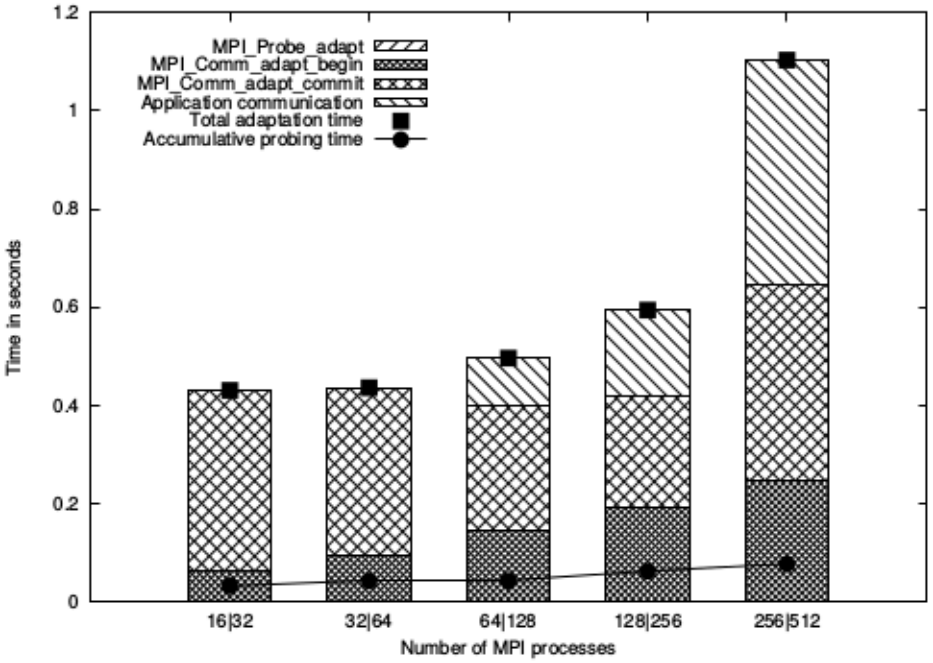
Elastic Surrogate Model Construction for a Statistical Inverse Problem

- 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

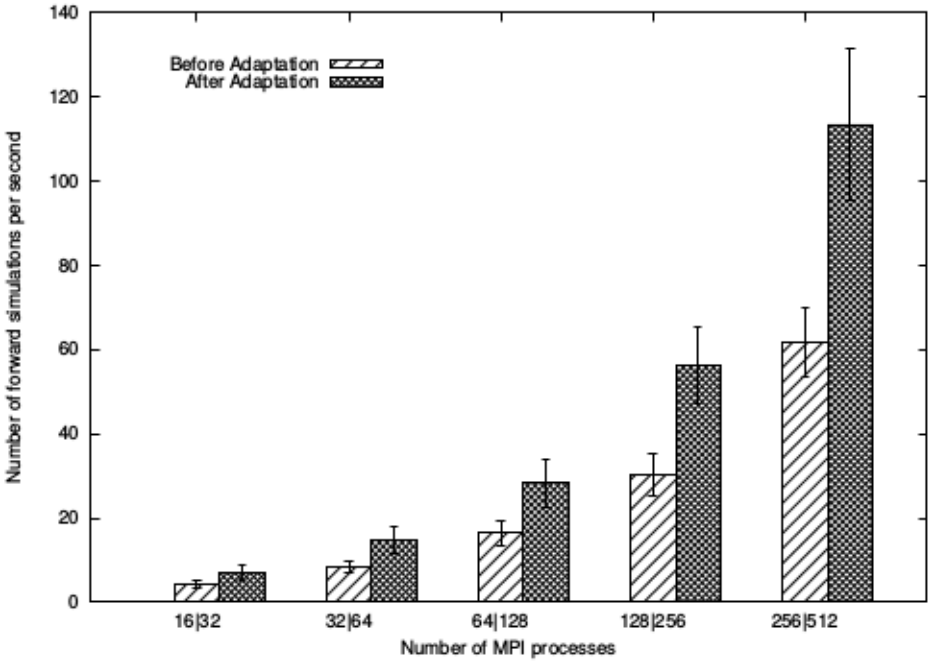


Performance Results

D3



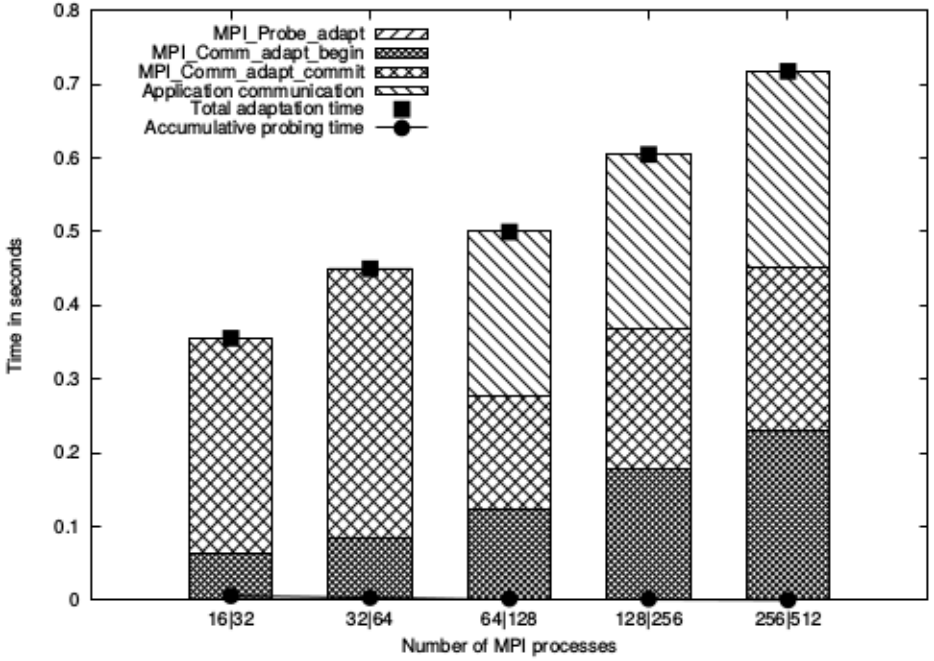
Adaptation overhead (small)



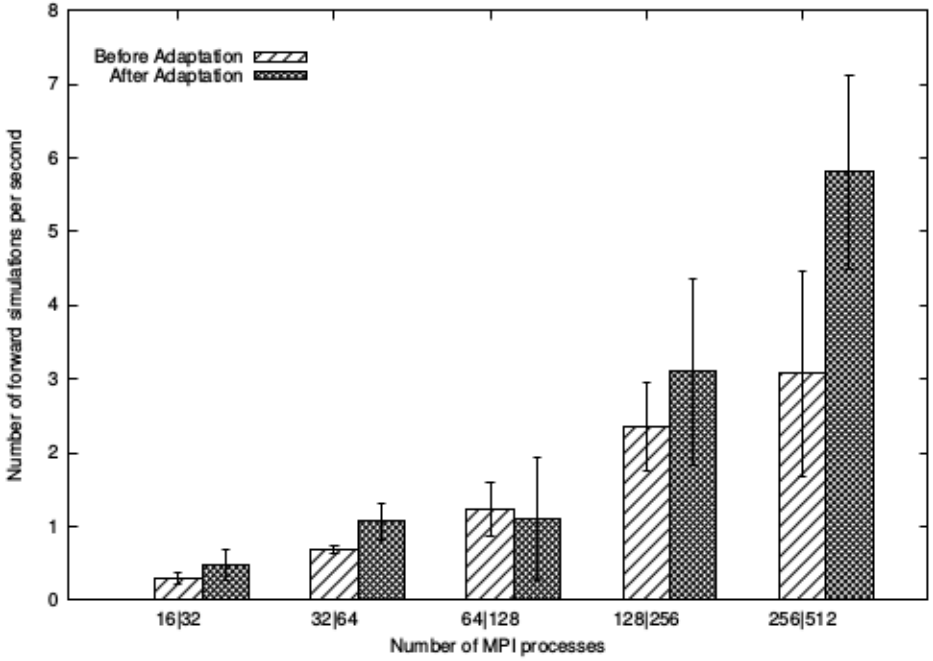
Performance change (small)

Performance Results

D3



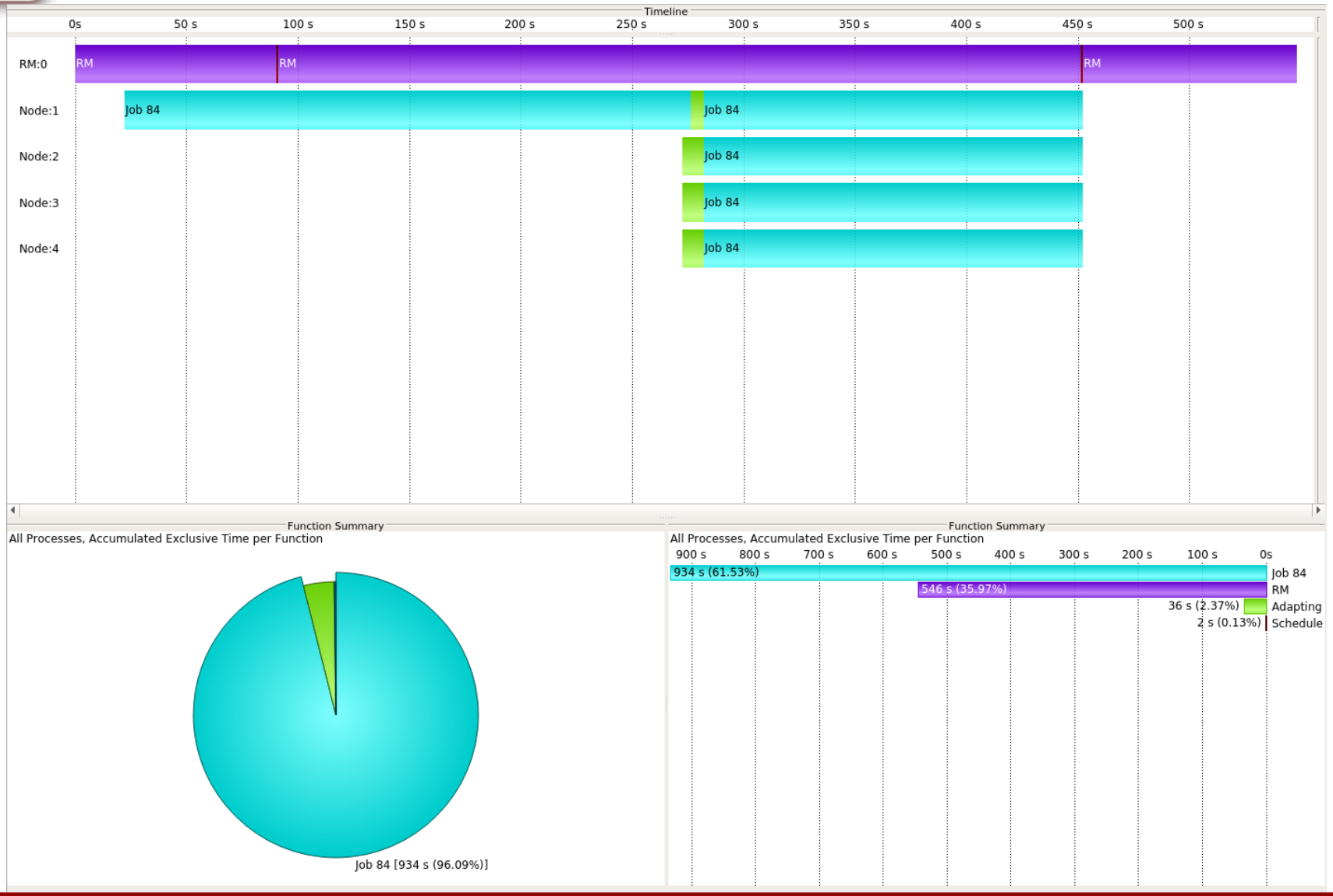
Adaptation overhead (large)



Performance change (large)

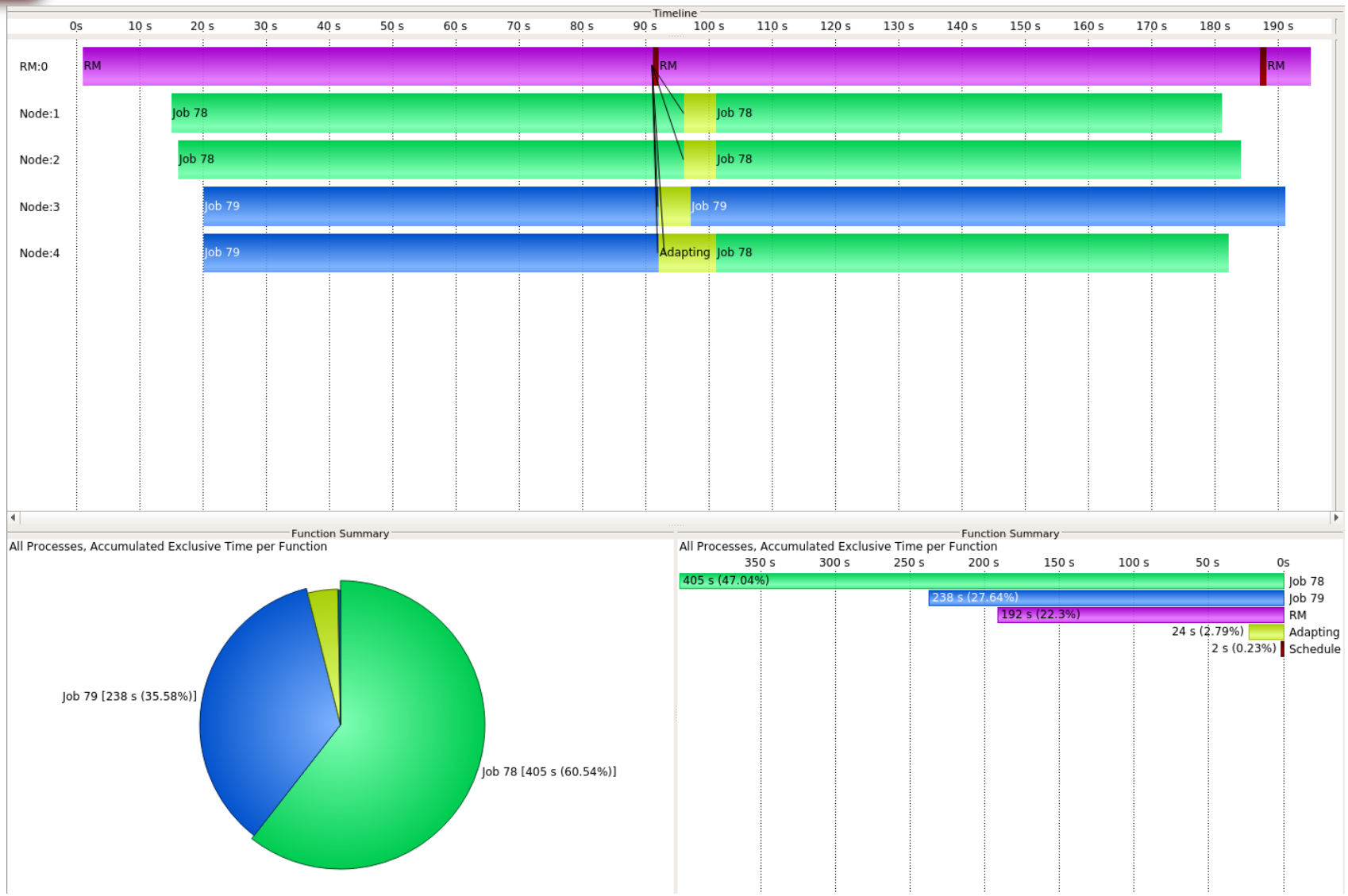
Adaptation Visualisation

D3



Adaptation Visualisation

D3



Conclusions and Future Work

D3

Proposed an extension to MPI that consists of 4 operations for:

- Initialization
- Probing for adaptations, and
- The creation of adaptation windows

Supporting these operations required changes to the resource manager and the PMI, in addition to the MPI library.

Our prototype implementation based on MPICH and SLURM provides:

- Satisfactory performance for our current use cases
- Latency hiding properties that minimize waiting times in preexisting application processes

Future work:

- Scheduling and tuning based on performance modeling
- Potential move to new resource manager (e.g. Flux) and PMI implementations (e.g. PMIx)