Malleability for HPC

ZIH-Colloquium | June 23, 2022 | Josef Weidendorfer

Includes Work by

Vincent Bode, Dai Yang, Amir Raoofy,
Tilman Küstner, Carsten Trinitis, Martin Schulz

Isaias Compress Urena, Michael Gerndt

The Role of the Future Computing Group at LRZ

User Requirements

Compute Demands | Ease of Use

Technology HW & SW

Cost-Effective | Sustainable/Green

Analyse, Predict

Future Computing Group

Recommend

Understand best options – not just t

Recommendations internally (for system pu
and externally (for supporting LRZ

Includes research on how to improve the operation and usage model, e.g. by adding **Malleability**

Outline

- Definition / Motivation
- LAIK: a runtime for elastic HPC codes
  - Overview
  - Examples
  - Directions
- Ongoing efforts

Definition "Malleable"

American Heritage Dictionary

Capable of being shaped or formed

For this talk (HPC context)

The property of a parallel code
to be able to take advantage of
changes in available compute resources
during run time

Benefits of Malleable Codes for Compute Centers

Improved usage model

- Quick job submission (for interactivity) by on-demand shrinking of running jobs
- Shrinking instead of killing running jobs if a high-priority jobs comes in

Better resource utilization

- Reduce fragmentation on a system (alternative to back-filling to use idle resources)
- Dynamic migration to more adequate resources when they become available

Better operation

- Dynamic migration to enable quick maintenance (similar to VMs)
- Ability to react on predicted node failures (”pro-active” fault tolerance)

Outline

- Definition / Motivation
- LAIK: a runtime for elastic HPC codes
  - Overview
  - Examples
  - Directions
- Ongoing efforts

Original Context

- Envelope: German research project on self-organizing systems (2017 – 2019)
  KIT, TUM, JGU Mainz, RWTH Aachen
- Pro-active fault tolerance
  - Retreat application from node before (predicted) failure
  - Application-integrated approach: needs support for malleability

envelope

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

Original Focus

- Enable “easy” porting of MPI codes for malleability

- Shrink / expand based on external requests
  - Process IDs (ranks) not integral part of API
    - Library controls number of processes, knows how to distribute data (application specifies partitioner algorithm)
    - Constrain to iterative codes using “owner computes” paradigm
  - Automatic re-partitioning on resource size changes

- Enable fault tolerance
  - pro-active (react to predicted failure in the future)
  - reactive (cope with spontaneous failure)

Basic Idea

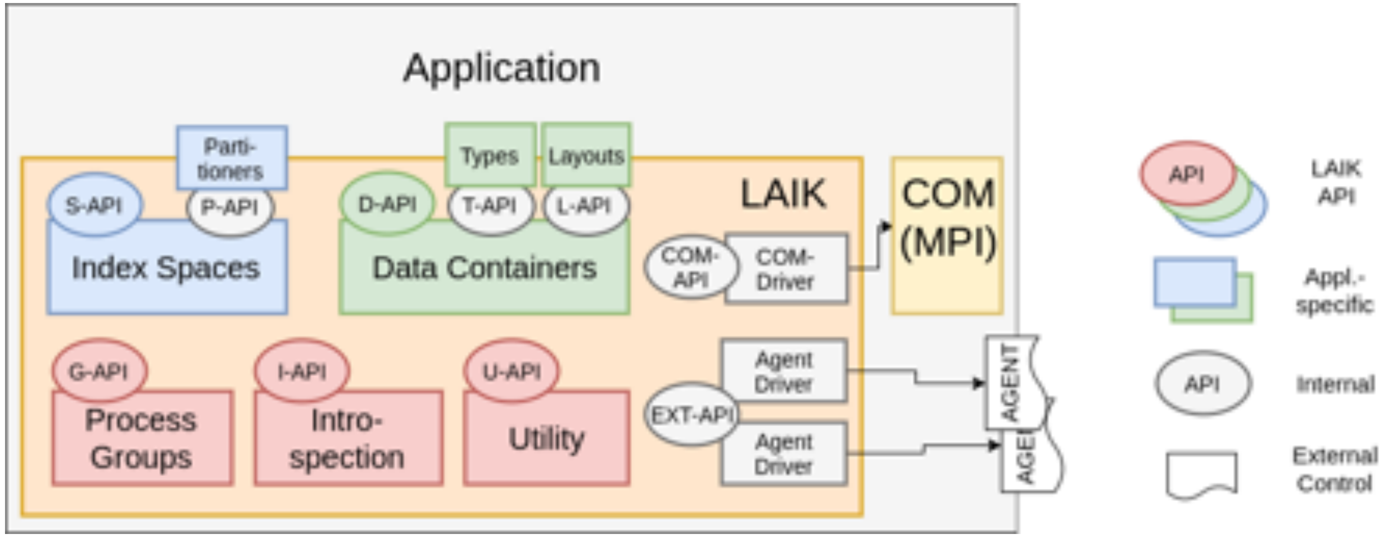
- Data is stored in multiple global (multi-dimensional) arrays, distributed over processes
  - Processes declare local access wish for array elements for compute phases
  - Value changes are globally propagated on demand

- Any communication = value updates of array elements

- We want to specify the communication requirements in an **abstract** way
  - Not based on MPI ranks, but: what data needs to be locally available for computation
  - Requirements declared once ➔ calculate communication schedules
    - executed multiple times on demand

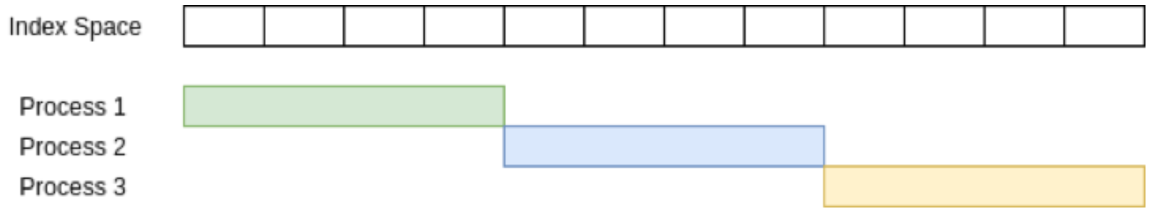
LAIK

- SPMD, can co-exist with MPI / OpenMP
  - a LAIK entity (process) is a Unix Process
- C API, runs on Linux & MacOS
- **github.com/envelope-project/laik**
- Communication backends as plugins: currently MPI, TCP

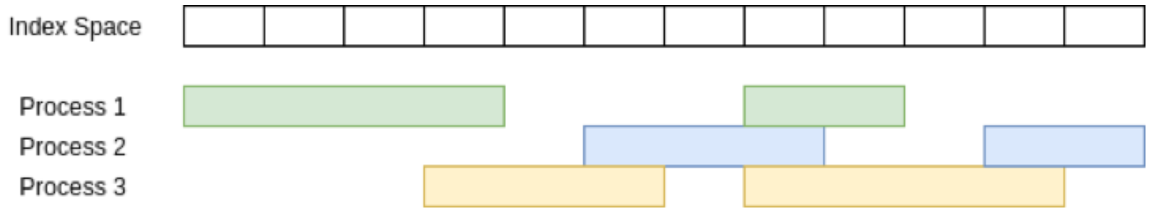


Index Spaces, Partitionings, Partitioners

- Example: disjunctive partitioning

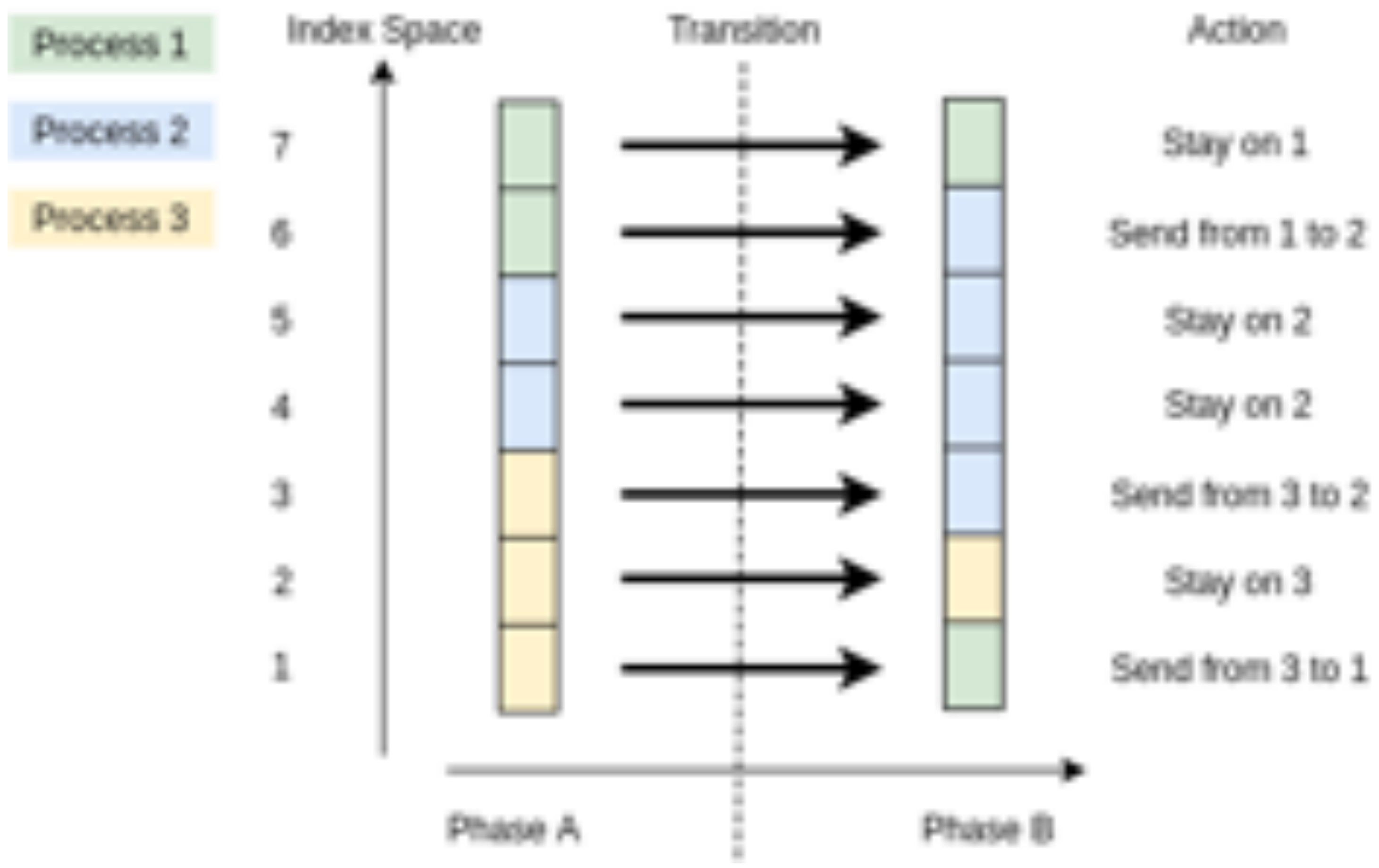


- General partitioning: multiple processes per index

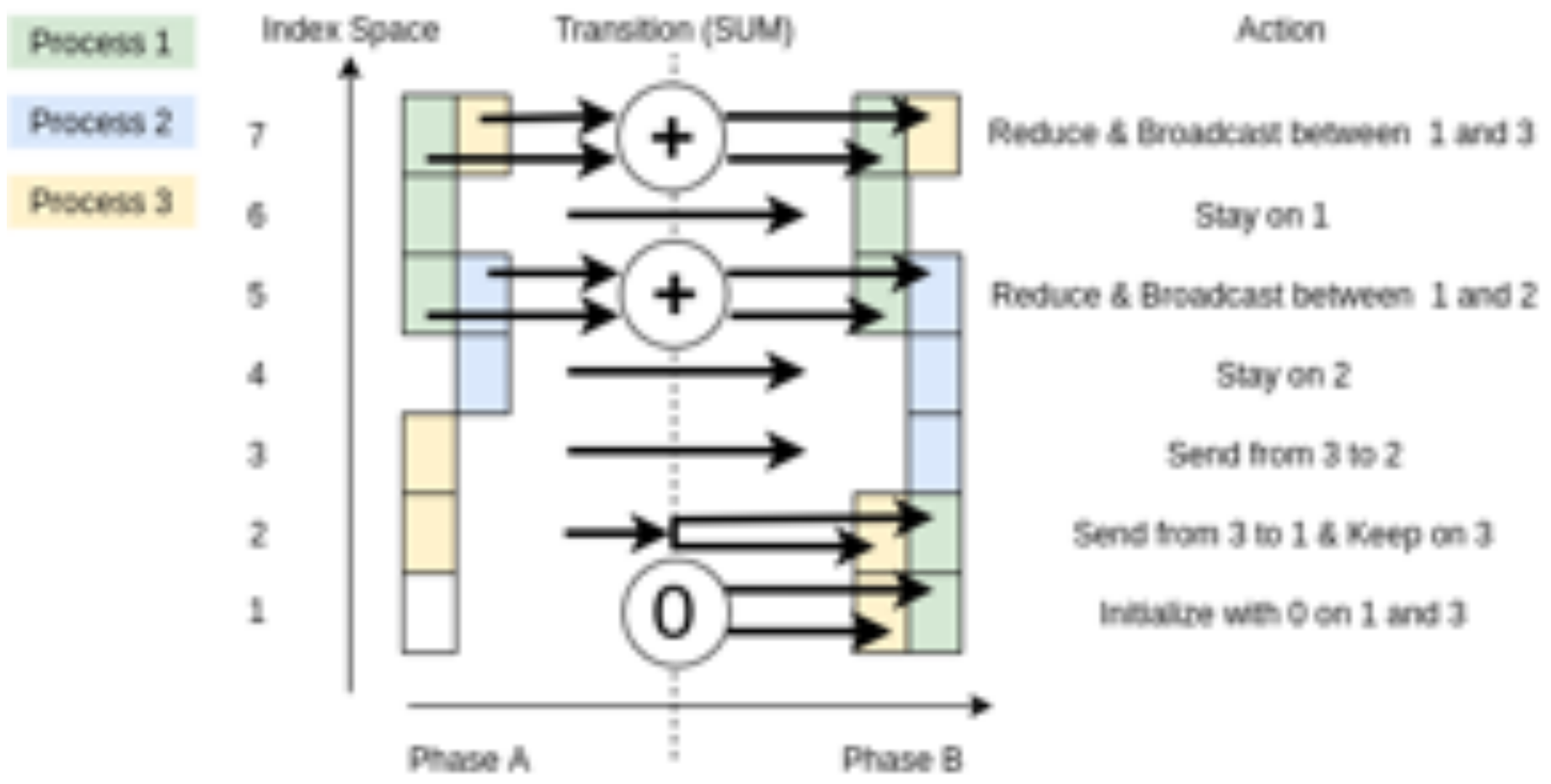


- Custom partitioner algorithms supported
  - can use another partitioning as input (e.g. with ghost layer)

Communication Schedule on Update, same Container



Communication Schedule on Update, same Container (2)



Needed: reduction operation (can be custom)

Data Containers

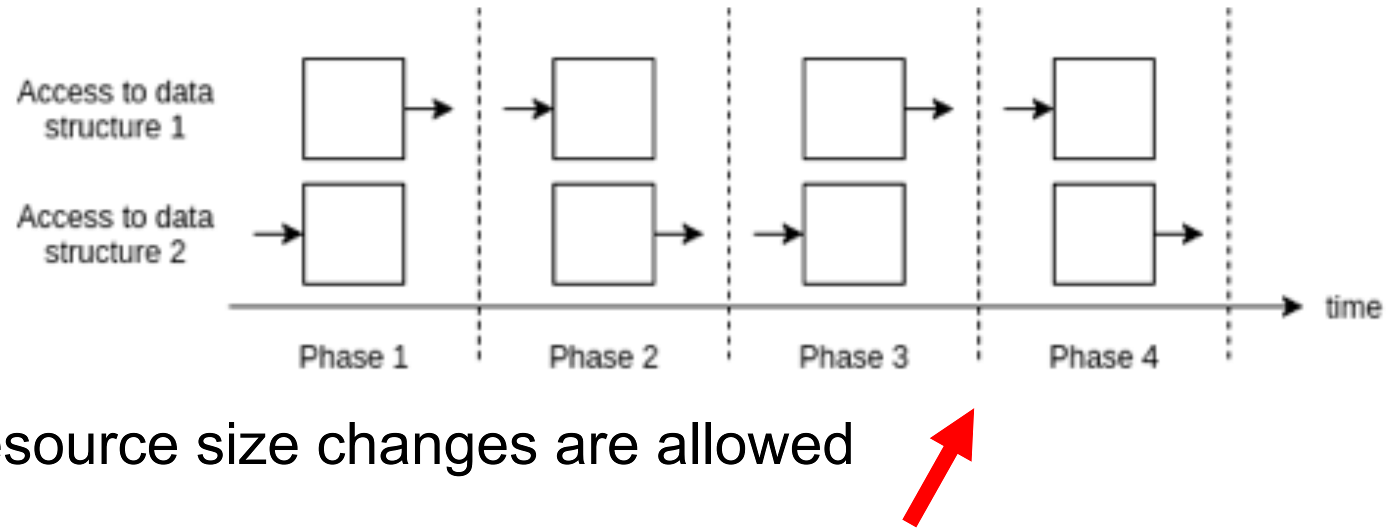
- Memory for data arrays, known to LAIK
- Same type for each index in a container
  - Type needed for reductions
- Layout: how are indexes locally ordered in memory
  - Application can specify custom layouts

- Explicit allocation
  - Specify set of partitionings to reserve space for, LAIK allocates
  - Get memory address for a local index (application knows layout)
  - Trigger Update: can only involve partitionings specified in allocation

- Repartitioning: allocate new, copy (= Update), free old

Application Structure (suggested, not enforced)

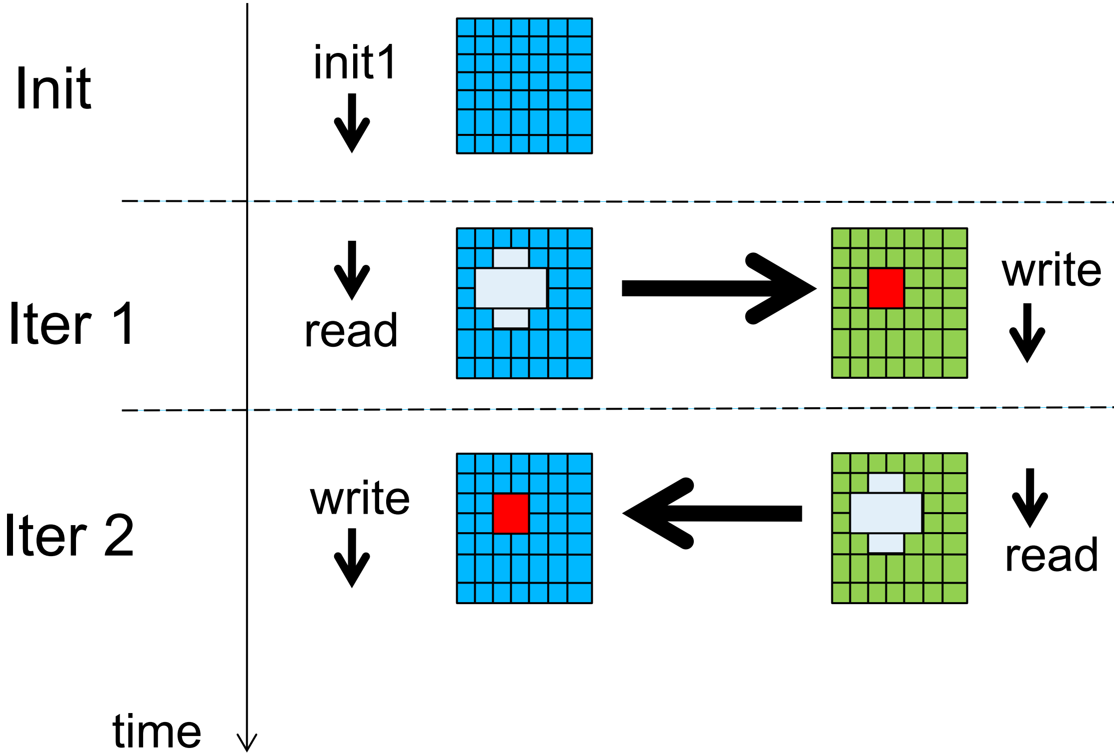
- Sequence of compute phases
  - Trigger adequate updates on phases changes



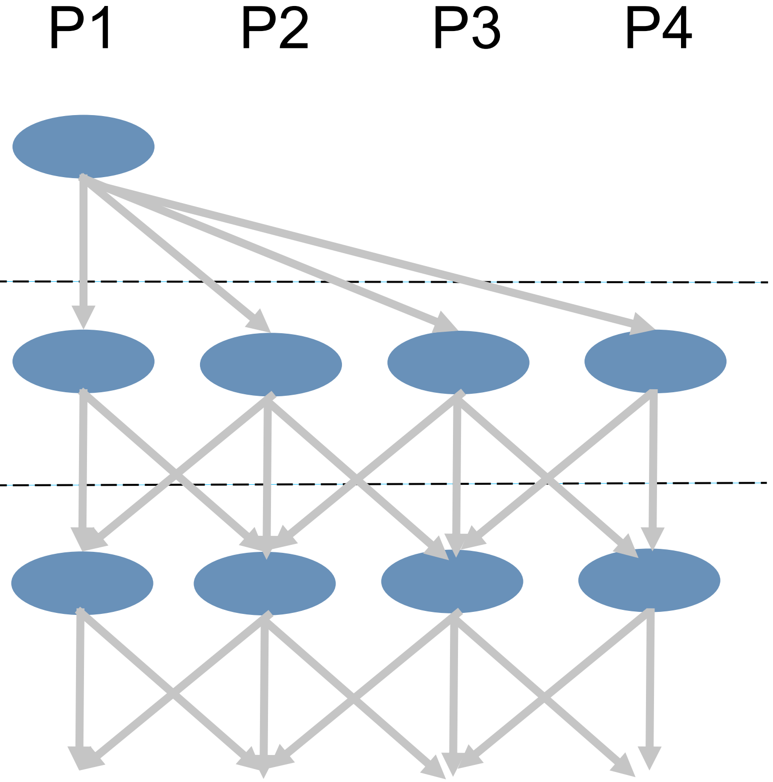
- Mark program points where resource size changes are allowed
  - Global synchronization
  - Results in re-runs of partitioner algorithms
    - Take joining / leaving processes into account
    - Data redistribution is “just” an update from old to new partitioning

Example: 2D Jacobi

- Two matrices, 5 point stencil
- Partitionings: “Init”, “Read with halos”, “Write” (disjunctive)
- Phases



Task Graph



LAIK Update Object

- Declaration
  - Before/after partitionings of same index space
  - Do update into same or different allocation?
  - Reduction operation
- Contains calculated communication schedule as instruction stream
  - Different abstractions possible: <RunUpdate>, <MPISend, …>
  - To be run by the communication backend (or stack of backends)
- Pre-optimize as much as possible
  - Can include code generation
  - Can be done off-line using pre-calculated partitionings in files (Metis runs)

LAIK Update Object (2)

Partial Specialization

- Information which may not be known at declaration time
  - backend to use, hardware available, machine topology
  - data container (from/to), type info, layout, memory addresses
- whenever new info becomes available, run optimization
  - can be done locally, as long as all LAIK processes agree on basics

LAIK Update Object (3)

Benefits of instruction stream approach

- Optimization passes are decoupled code
- Application-specific passes possible
- easy selection via external configuration
- Instructions can be backend-specific (backend executes instruction stream)
  - Enables layers of backends, which pass instruction stream down
    e.g. Smart NIC-support: reduction offloading, resource reservation
- Variants of instruction streams to react dynamically
  - Change of job priority from job scheduler

LAIK Update Object (4): Optimization Passes

Examples

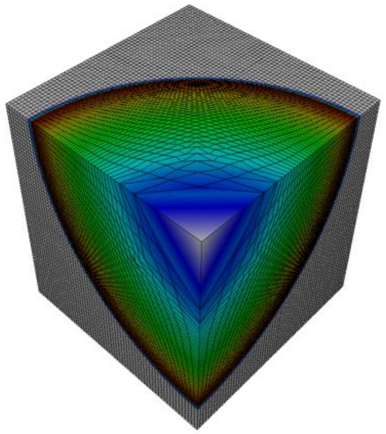
- Lower abstraction:
  “send indexes 1-5 from container X” → “send 5 ints at 0xX”
- Merge multiple sends between same peers into one send
- Copy pieces into pre-allocated network buffers
- Different reduction algorithms depending on size / topology
- Detect Pattern & replace with MPI collective (All-to-All)

Provided functionality for Passes

- Can create backend-specific calls to early resource alloc/free
- Reordering via attaching ordering labels

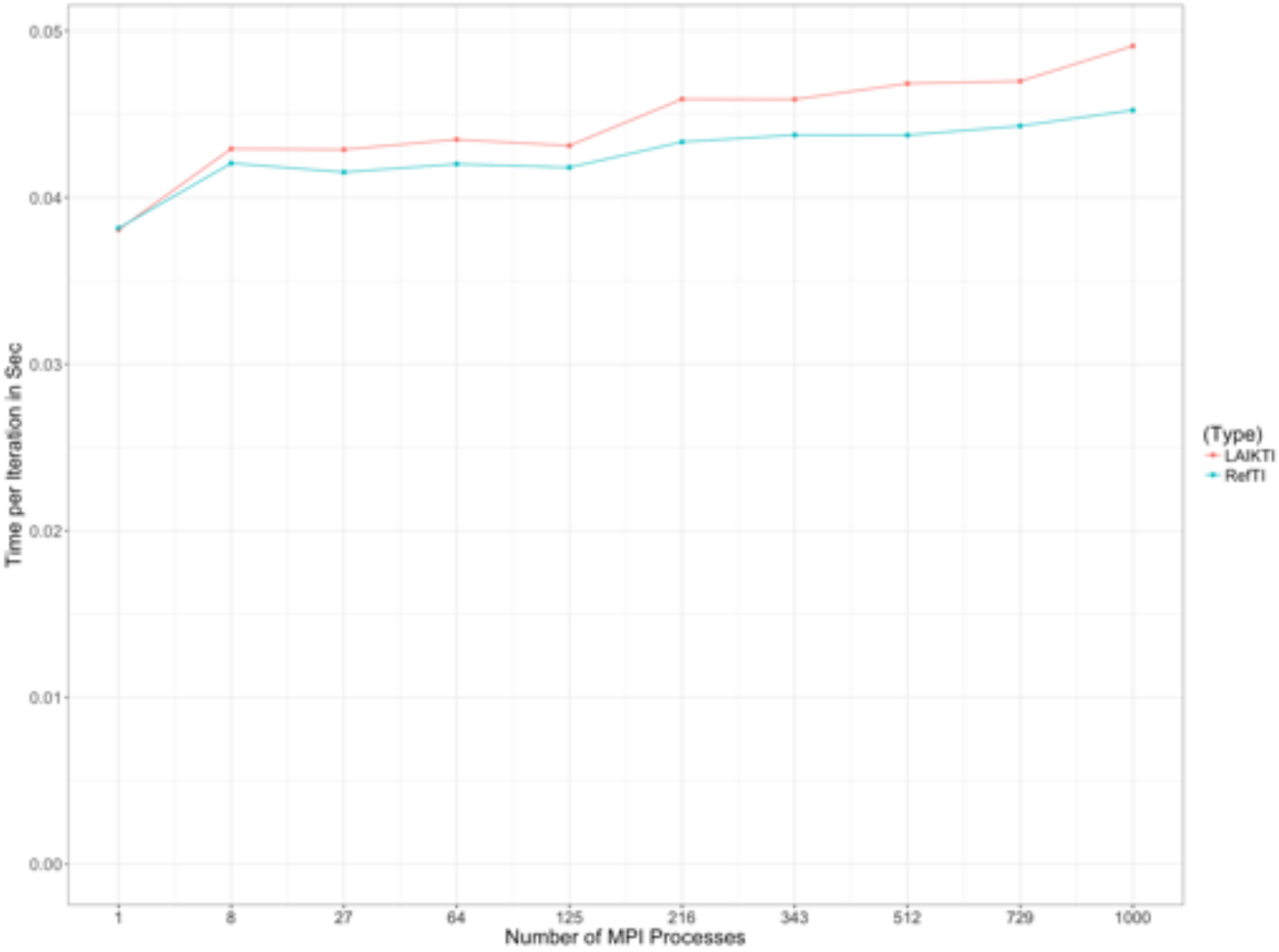
Porting of LULESH (Shock Hydrodynamics)

- Incremental porting from MPI code
  - Small steps, easy to check for correctness
  - Eliminate 2800 lines of comm code (~50%)
  - Almost no changes in main loop of reference code
- Performance/scalability similar to MPI
- Added elasticity and external control



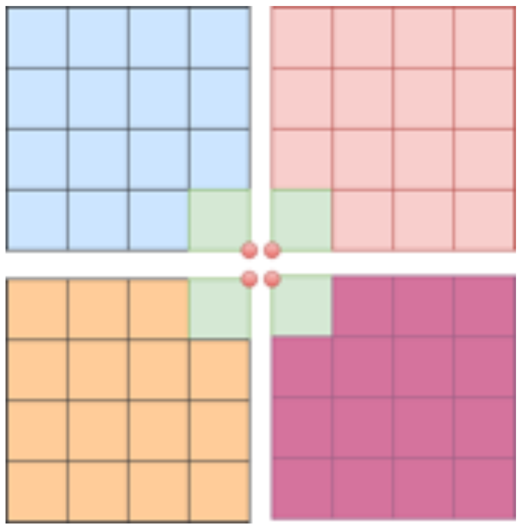
LULESH: Weak Scaling

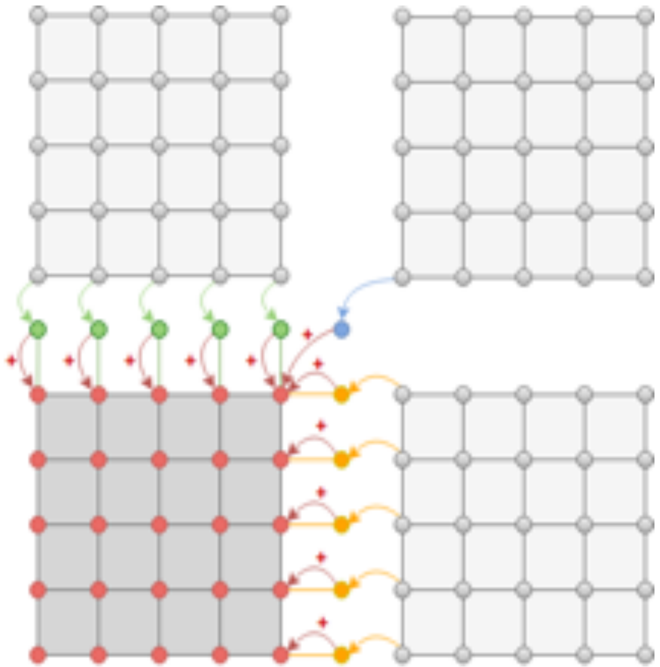
- Normalized time per Iteration



MPI Communication for Kernels

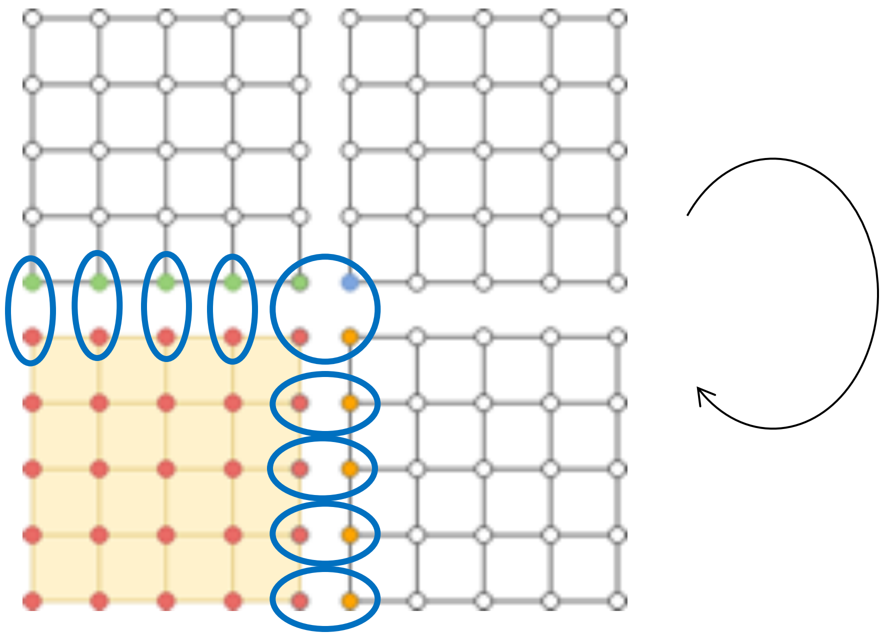
- Kernel 2: communicate neighbors’ corner data, and do local reduction (sum)





LAIK Model: Just Update

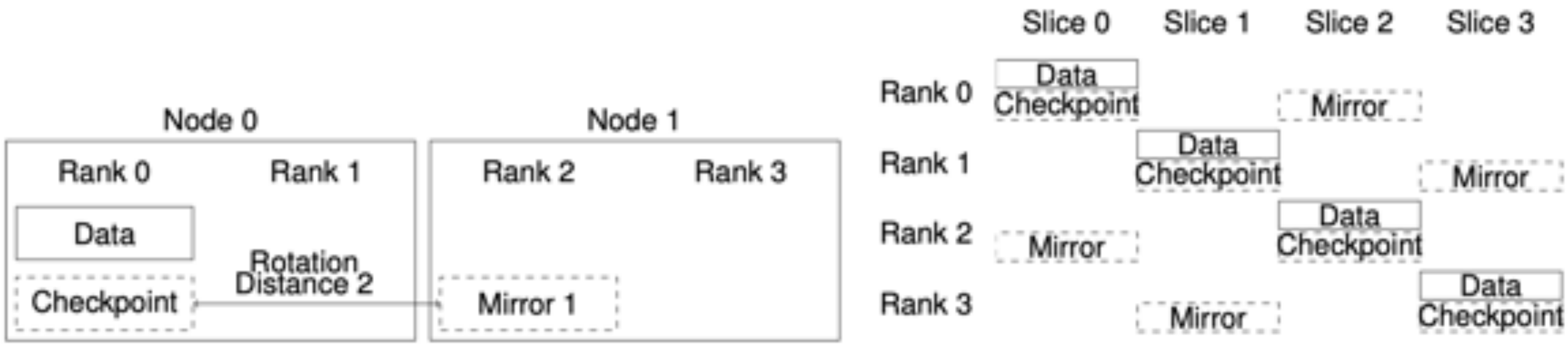
- Kernel 2: one partitioning with overlapping access at edges: triggers reduction on update



use SUM
reduction

Fault Tolerance for spontaneous Failures

Regular “in-memory” checkpointing, to neighbor node
Extend partitioning for redundancy, copy into “checkpoint” containers



Fault Tolerance for spontaneous Failures

1) Regular “in-memory” checkpointing, to neighbor node
   Extend partitioning for redundancy, copy into “checkpoint” containers

2) Node failure: Transition execution only partially completed
   - Values do not get updated correctly
   - Still, application can proceed (just “wrong values”)
     - Failure handling can be delayed to later point
       (or skips computation)

Fault Tolerance for spontaneous Failures

1) Regular “in-memory” checkpointing, to neighbor node
   Extend partitioning for redundancy, copy into “checkpoint” containers

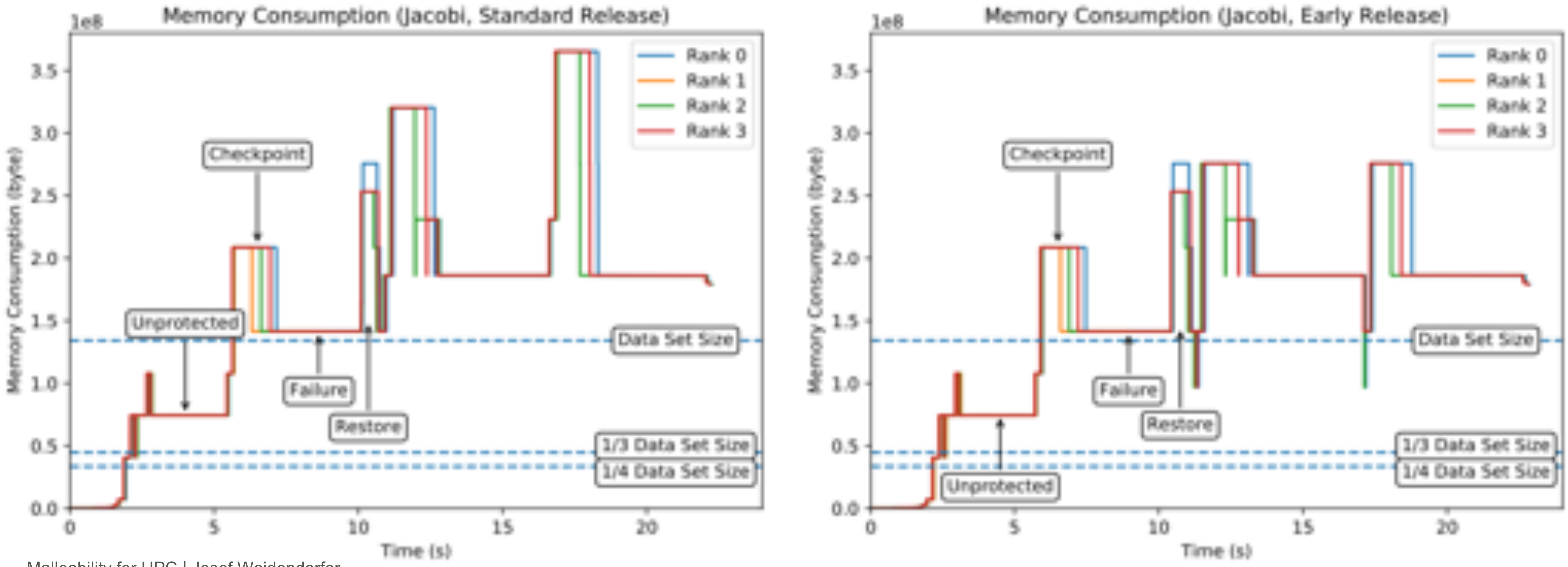
2) Node failure: Transition execution only partially completed
   Failure handling can be delayed to later point

3) Rollback to previous checkpoint
   Copy back, only working if redundancy was enough
   - Without failed processes, data distribution may be different, but can proceed
   - Eventually with Re-balancing step

Evaluated via (1) “TCP backend”, (2) ULFM MPI implementation

Memory Consumption

- ”Standard Release”: delete old checkpoint only after next checkpoint successful
- “Early Release”: delete old checkpoint before creating next checkpoint (higher risk!)



At SC19

© Eva Gellner

Other Directions LAIK

- Automatic load balancing
- Alternative for virtual topology
  - LAIK knows communication matrix for updates
  - Developer can specify which updates should be as fast as possible
- Other communication backends
  - Shared memory: 2-copy → 1-copy → 0-copy
  - RDMA on libfabric (allows direct resource reservation): less handshakes, better asynchronity
- Show that API works with requirements from real-world applications
  - More ports

Understand what is needed in MPI to best support these features

Outline

- Definition / Motivation
- LAIK: a runtime for elastic HPC codes
  - Overview
  - Examples
  - Directions
- Ongoing efforts

Efforts

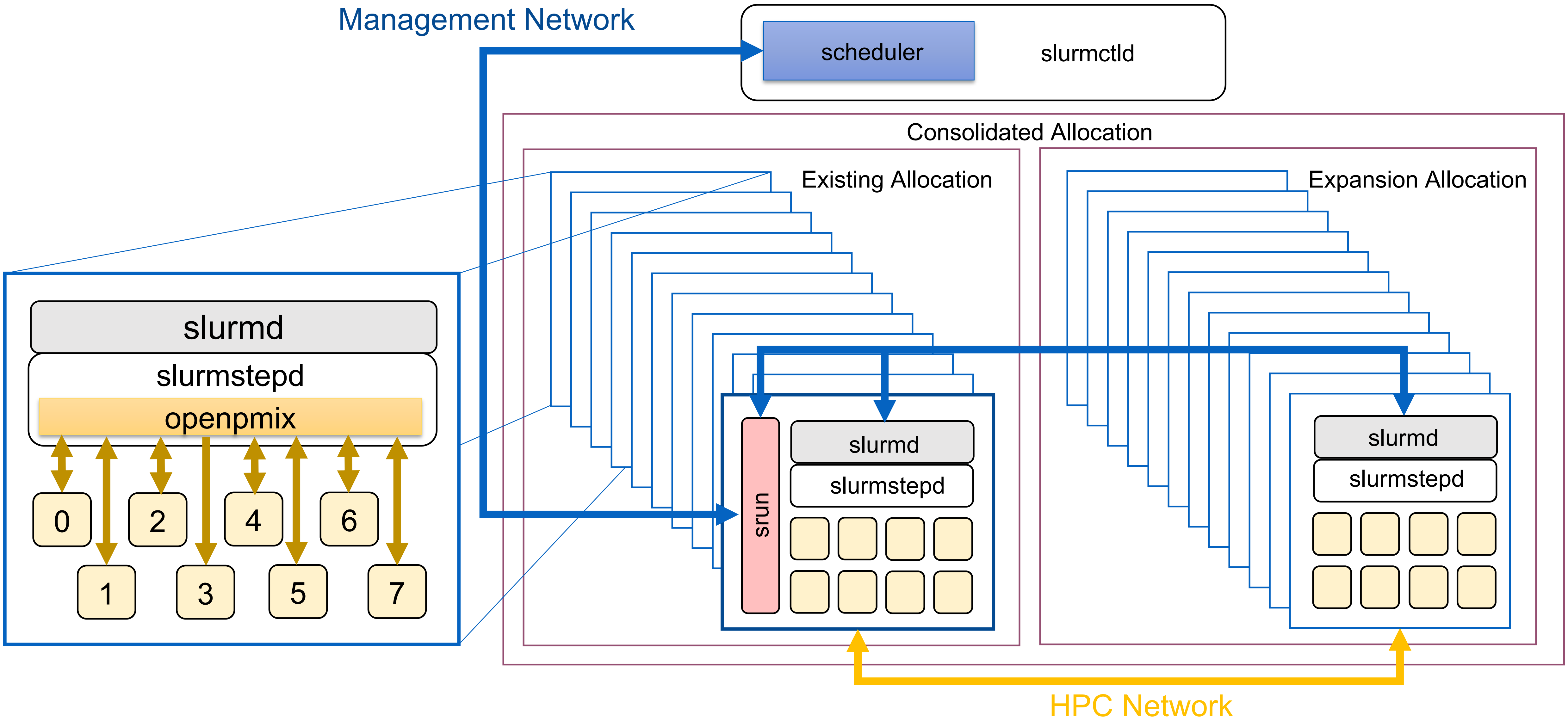
MPI

- TRR Invasic / EU DEEP-SEA
  Research on extending MPI for elasticity, proposal to MPI forum
  (by Isaias Compress Urena, Michael Gerndt)
- Ongoing discussion in MPI WG Malleability
  - Very flexible, covers different use cases

PMIx

- Updates for malleability
- Pushed by work in EU DEEP-SEA

Process Interactions on Malleable Allocations



D3

Adaptation Step 1

D3

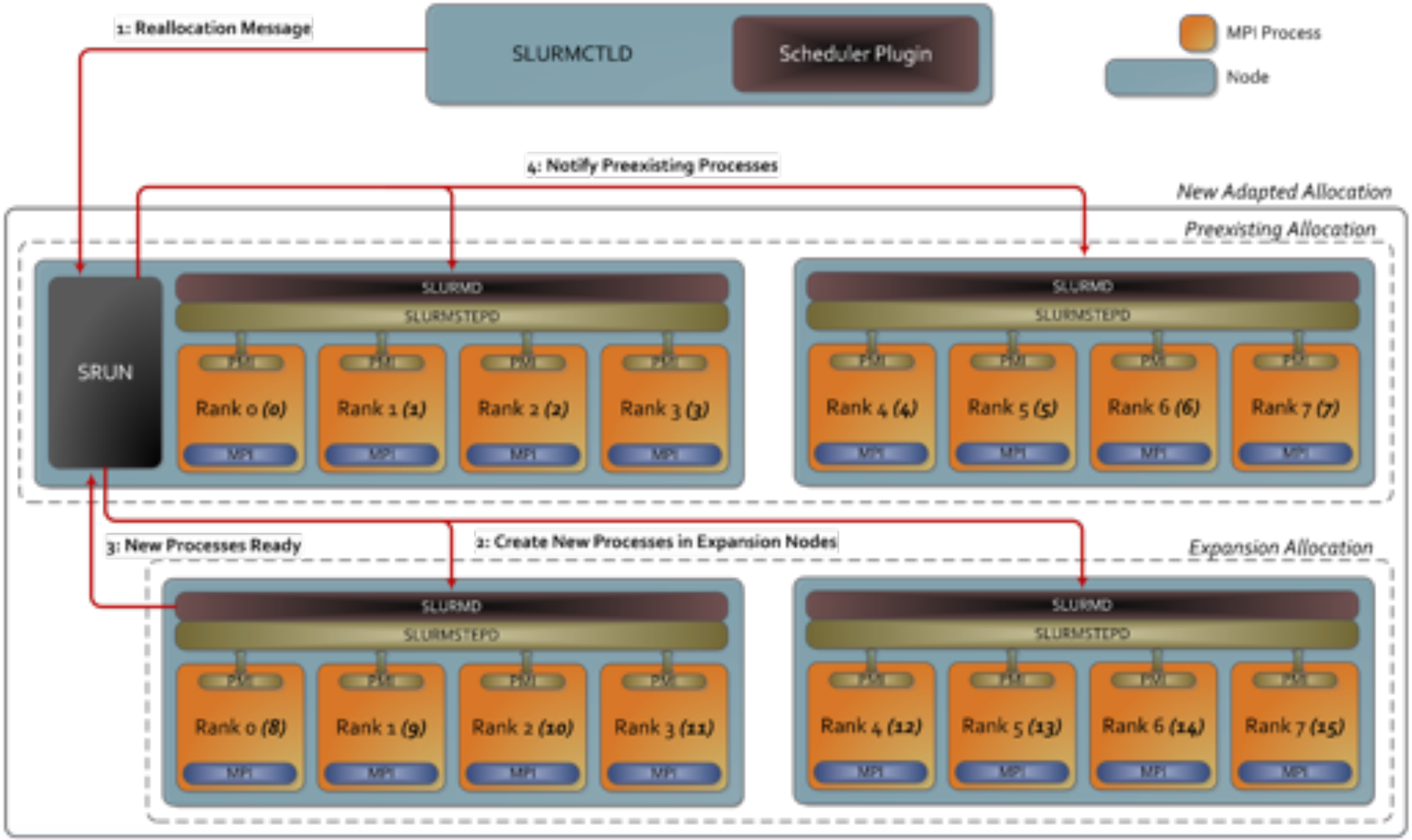
Adaptation Step 2

D3

Adaptation Step 3

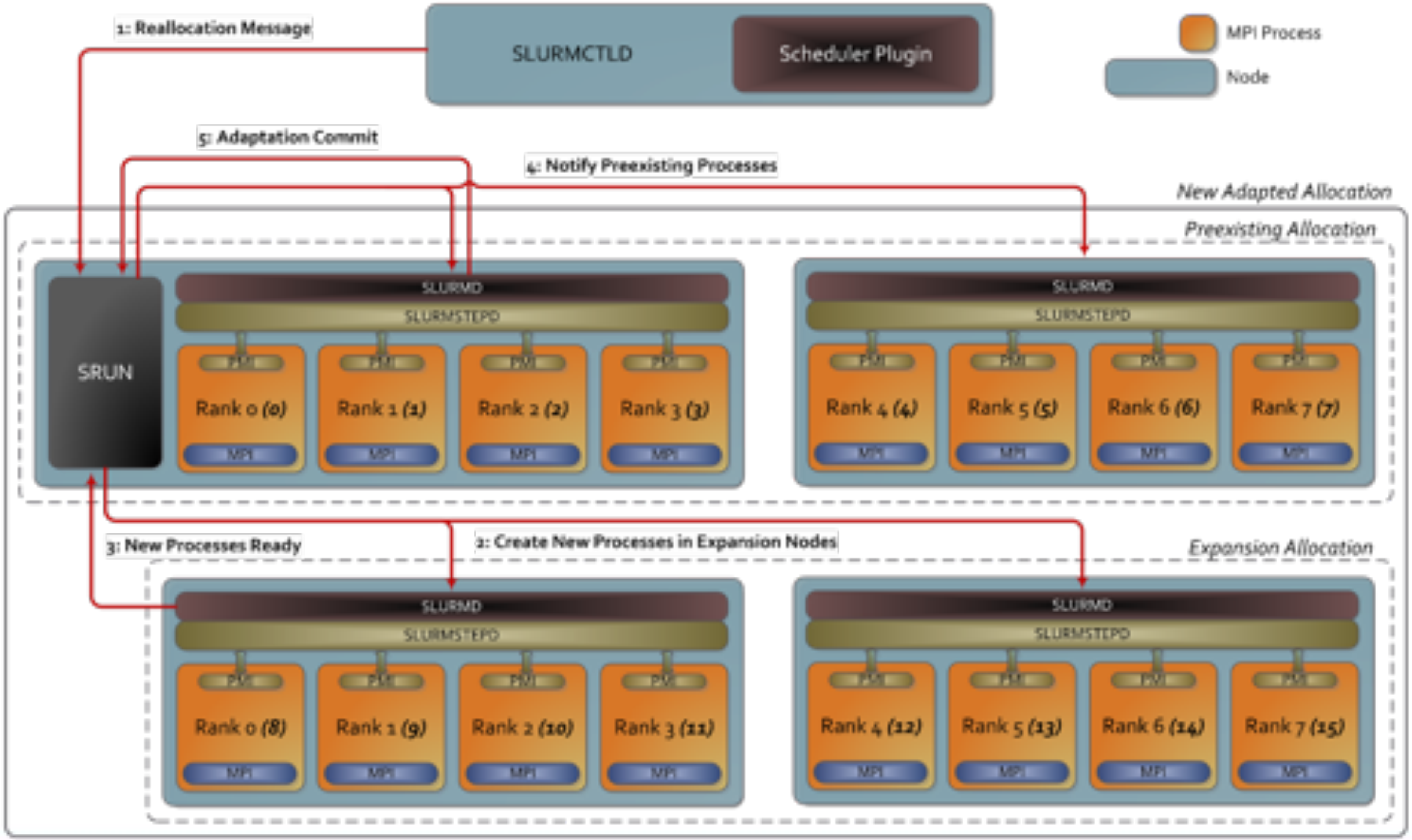
Adaptation Step 4

D3



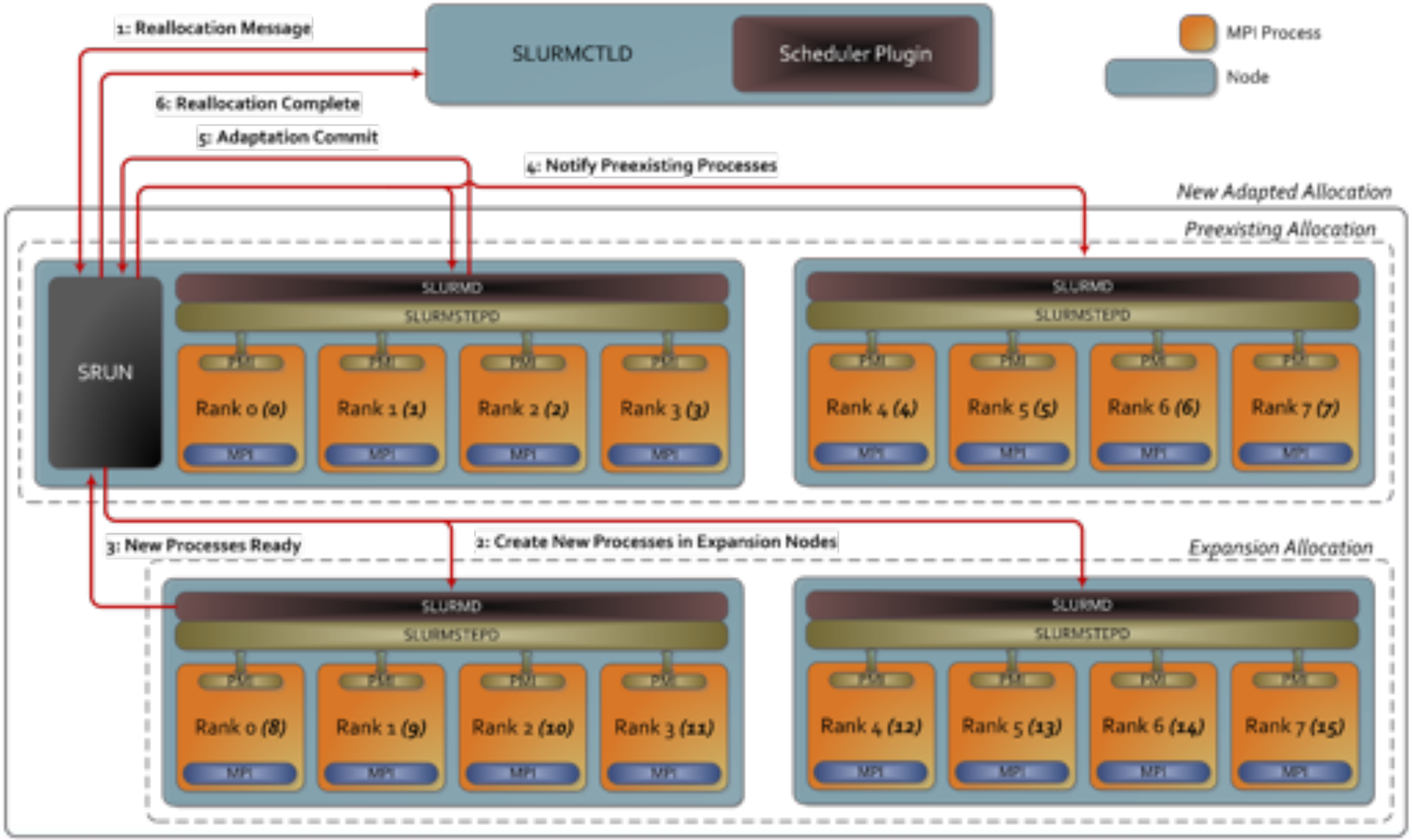
D3

Adaptation Step 5



D3

Adaptation Step 6



Efforts

MPI

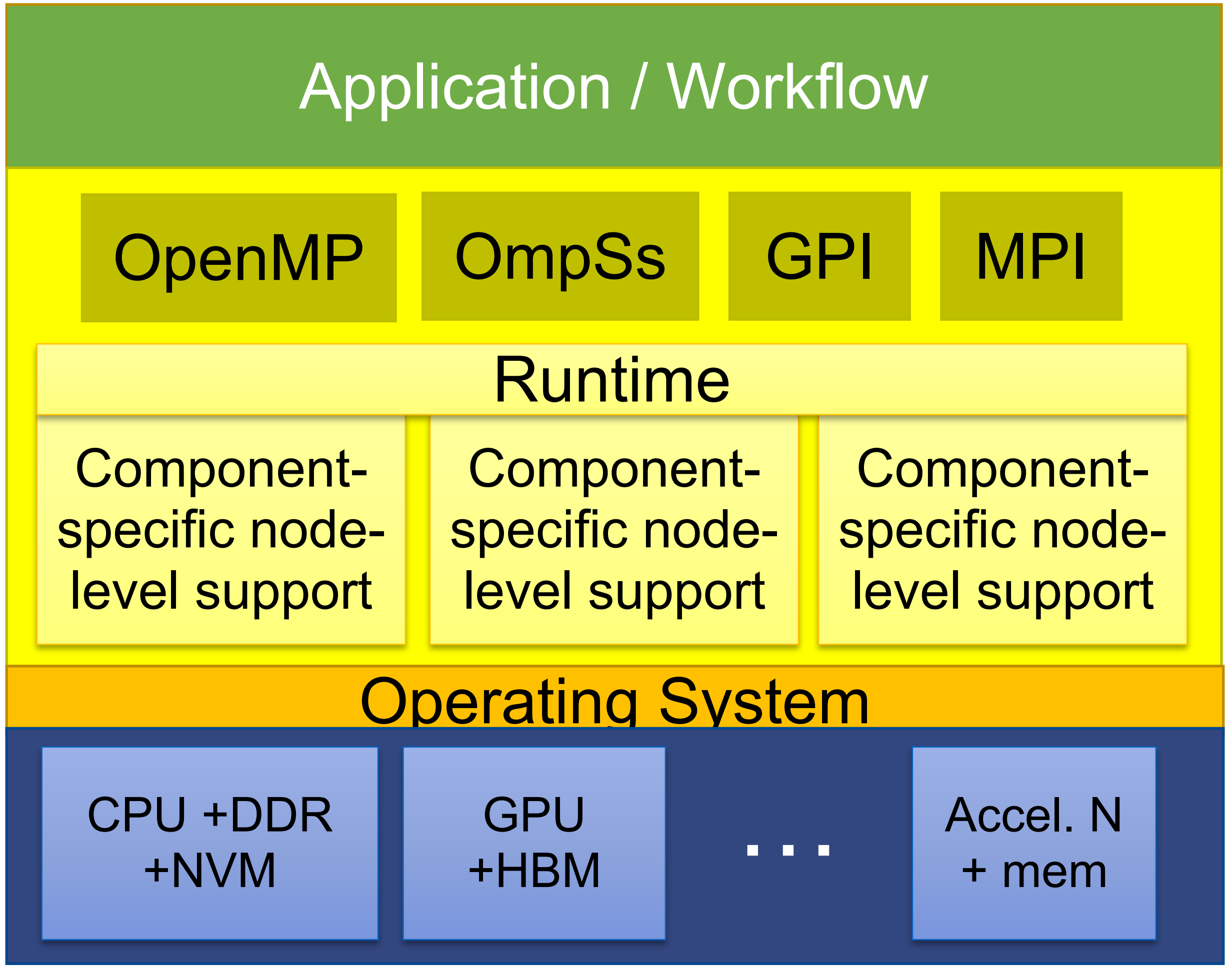
- TRR Invasic
  Research on extending MPI for elasticity, proposal to MPI forum
  (by Isaias Compress Urena, Michael Gerndt)
- Ongoing discussion in MPI WG Malleability
  - Very flexible, covers different use cases

PMIx

- Updates for malleability
- Pushed by work in EU DEEP-SEA

PMIx in DEEP-SEA System Software

- PMIx enables interoperability
  - Programming models:
    - *MPI, GPI2, GPI-Space, OmpSS*
  - Tools support:
    - *Trace analyzers, memory analyzers, monitors*
  - Network endpoints setup:
    - *Tools over management network*
    - *HPC network*
  - Heterogeneous task mappings
- Integration primarily with Slurm



lrz

Leibniz Supercomputing Centre

of the Bavarian Academy of Sciences and Humanities