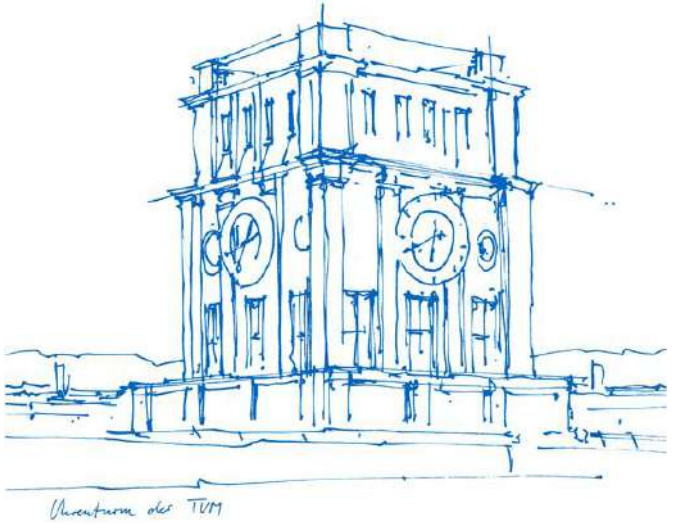
Design Principles of Dynamic Resource Management in High-Performance Parallel Programming Models

Dominik Huber[1], Martin Schreiber[2], Martin Schulz[1], Howard Pritchard[3], Daniel Holmes[4]

[1]Technical University of Munich, [2]Université Grenoble Alpes

[3]Los Alamos National Laboratory, [4]Intel Corporation



4th EuroHPC Workshop on Dynamic Resources in HPC

Dresden, 25.08.2025

OUTLINE

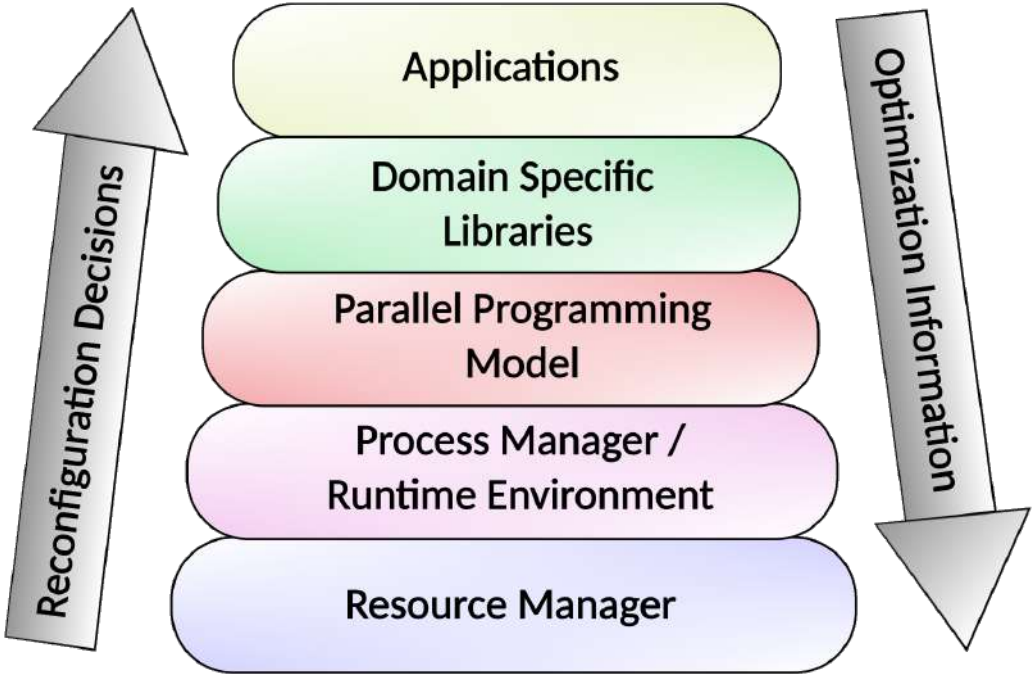
The Need for Common Interfaces

Design Principles

Minimal Interfaces

Prototype & Early Experiences

Dynamic Resources in the HPC Software Stack



The Need for Common Interfaces

State-of-the-art

- Specialized solutions
- Not portable/generalizable
- No standardization

Future?

- Common interfaces
- Specialization on top
- Standardization

OUTLINE

The Need for Common Interfaces

Design Principles

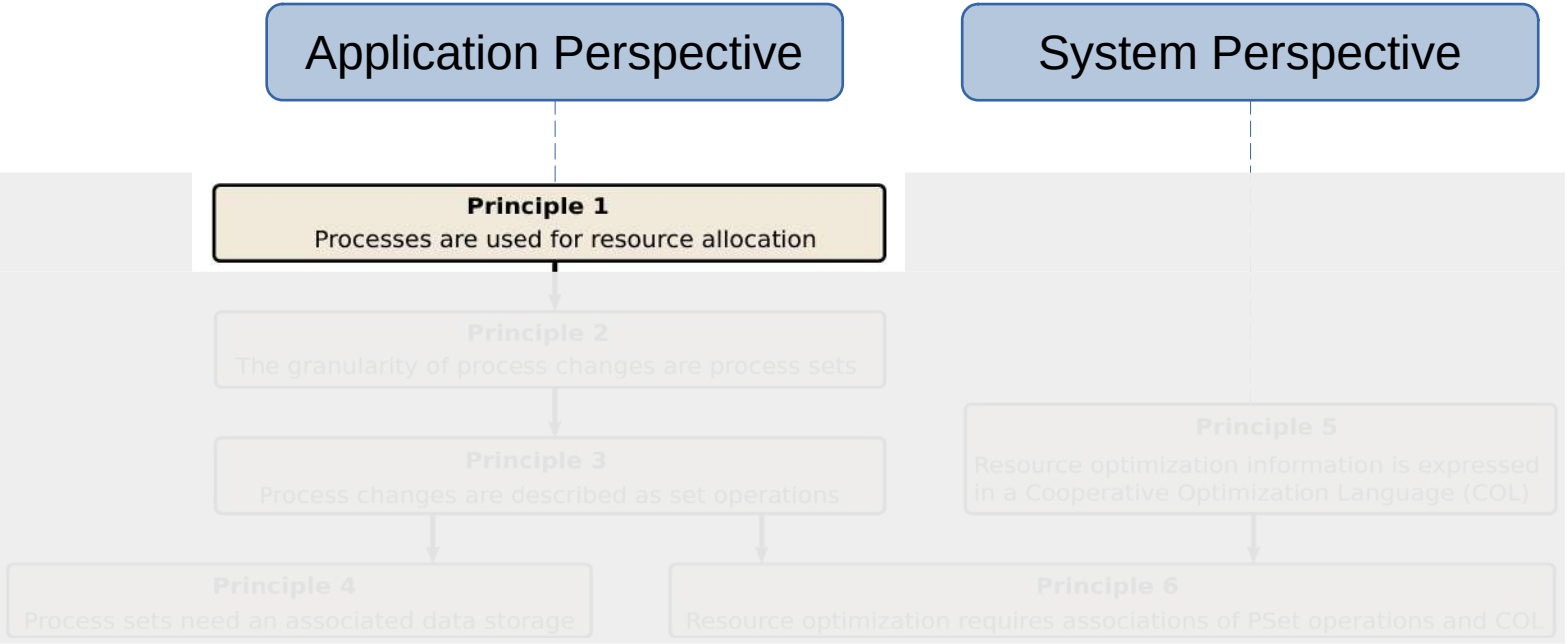
Minimal Interfaces

Prototype & Early Experiences

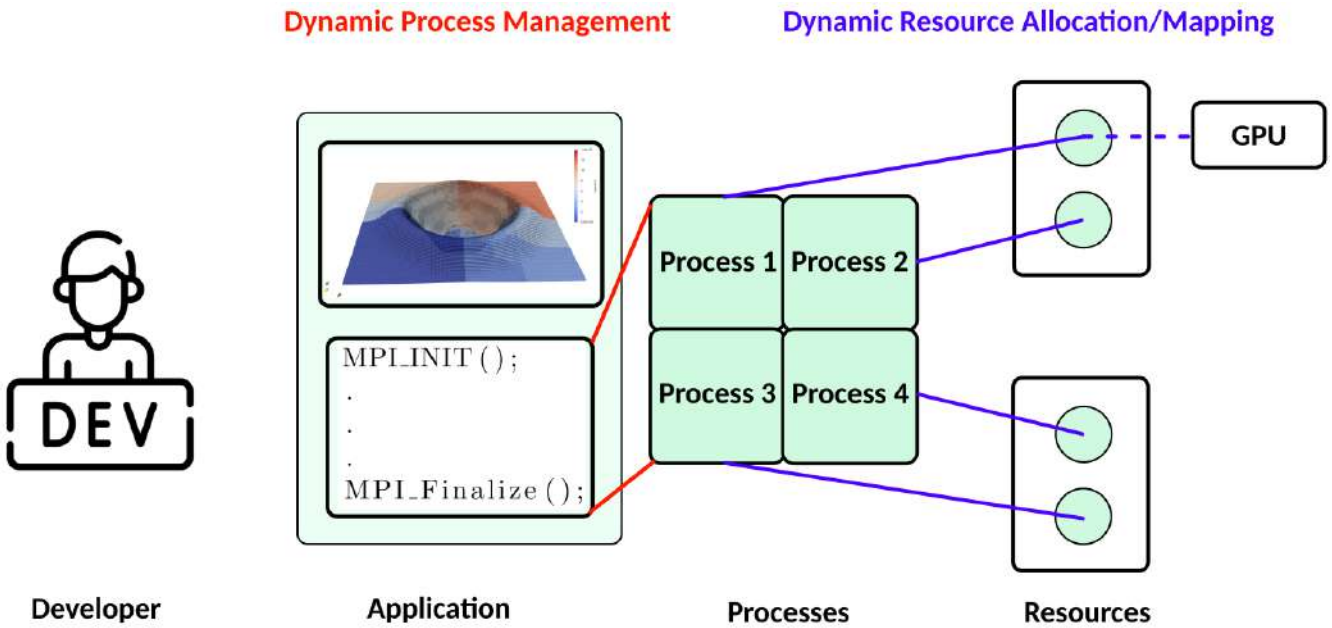
Principle-Driven Design

Common design principles for DRM mechanisms in the middelware stack

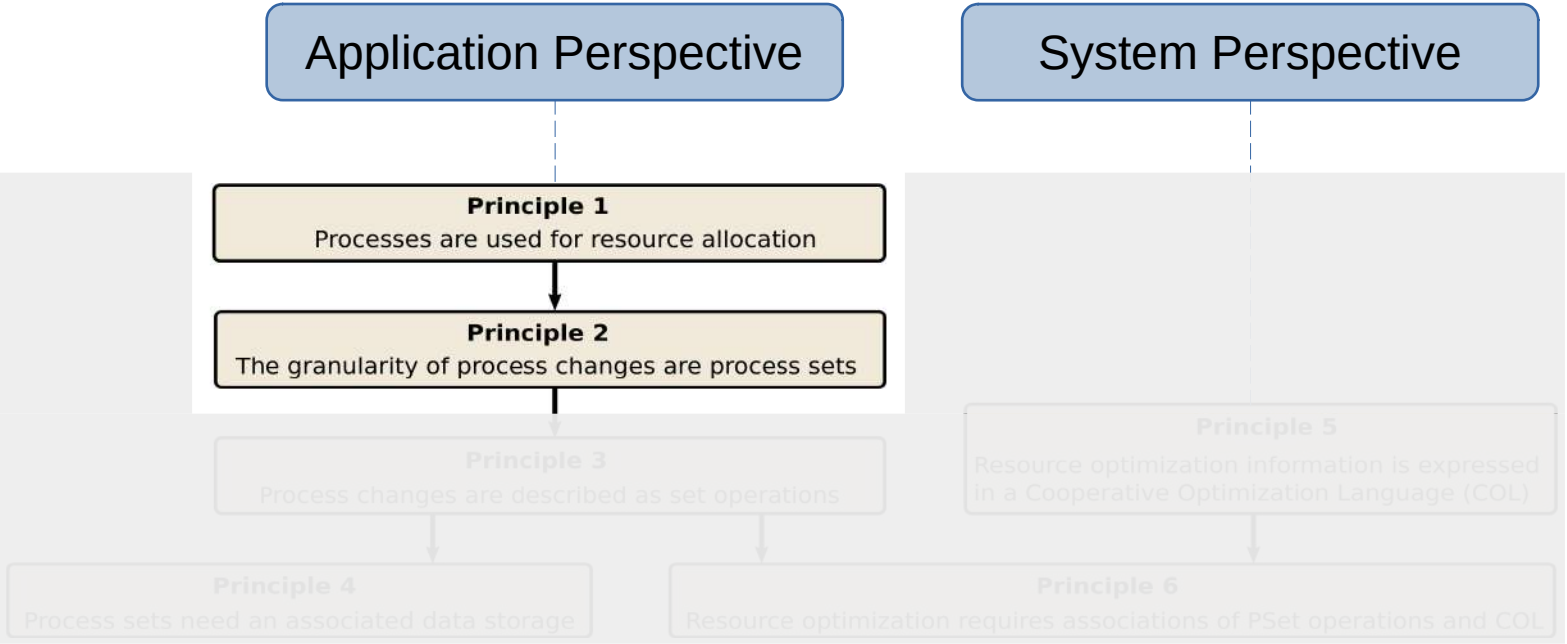
Principle-Driven Design



Principle 1: Processes are used for resource allocation

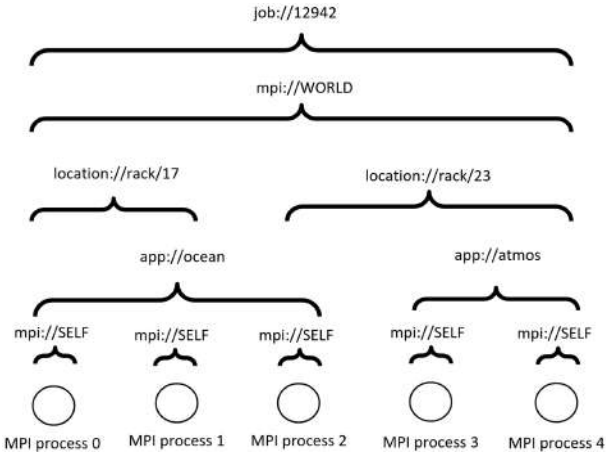


Principle-Driven Design



Principle 2: The granularity of process changes are PSets

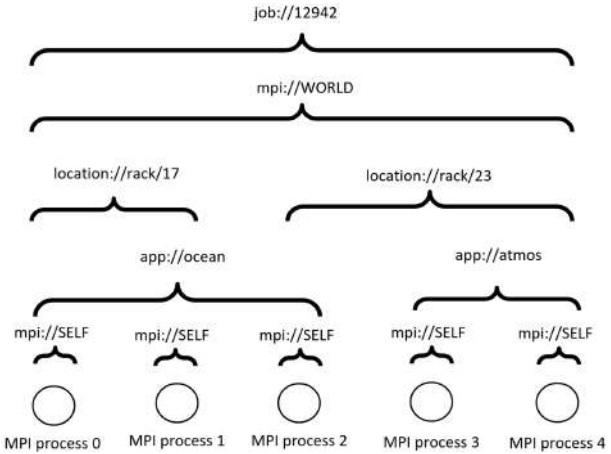
Definition “Process Set” (PSet): A PSet is a set of processes where both, the set and its members are uniquely identifiable. A **“0-Pset”** is an empty Pset which always exists.
Definition “Process Changes”: A process change is a change of association between processes/PSets and HPC resources.



Example of PSets in MPI (MPI: A Message-Passing Interface Standard Version 4.0. p. 504)

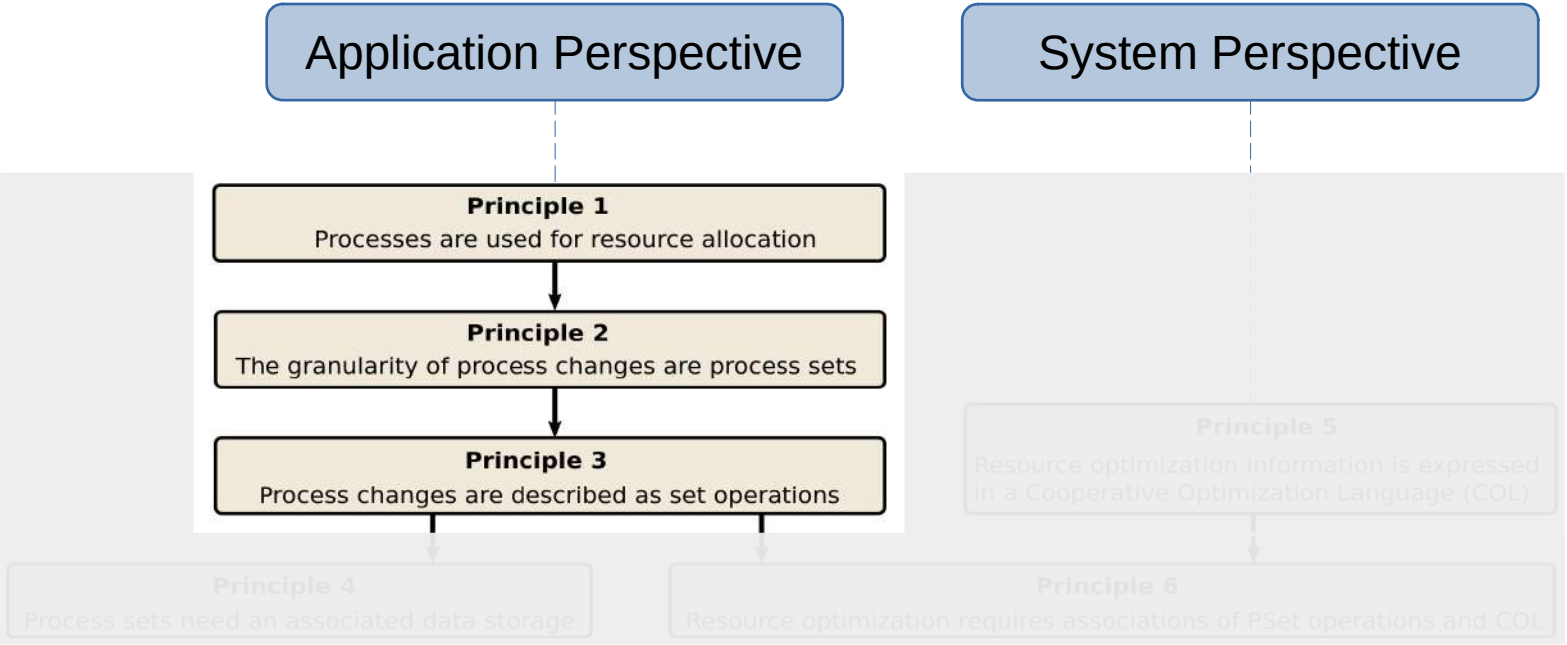
Principle 2: The granularity of process changes are PSets

1. PSets provide **flexible granularity**
2. Unique PSet labels allow **reference without listing members**
3. Unique Proc labels are required for **orderability and set operations**



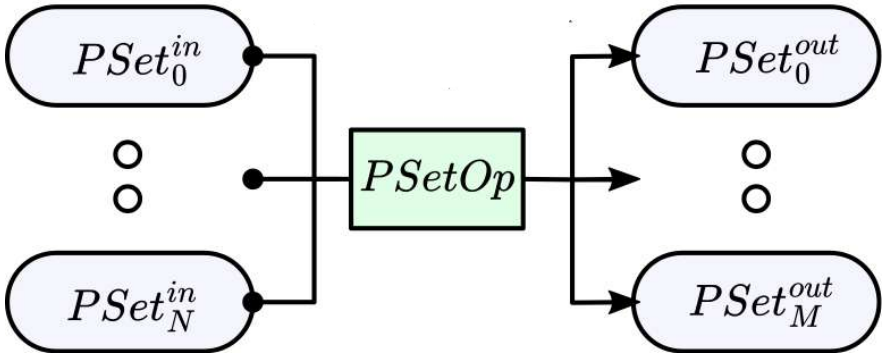
Example of PSets in MPI (MPI: A Message-Passing Interface Standard Version 4.0. p. 504)

Principle-Driven Design



Principle 3: Process changes are described as set operations

- **Definition “PSet Operation” (PSetOp):** A PSetOp is an operation on a list of **input PSets** that produces a list of **output PSets** based on a **well-defined rule**.



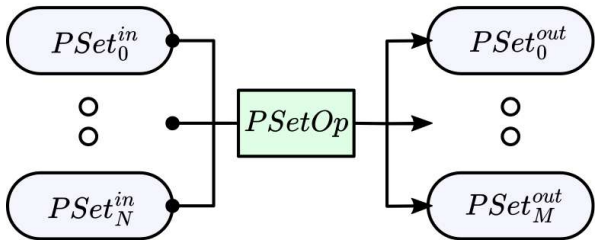
Principle 3: Process changes are described as set operations

- **Definition "PSet Operation" (PSetOp):** A PSetOp is an operation on a list of **input PSets** that produces a list of **output PSets** based on a **well-defined rule**.

ADD: $P^0_{in} \to P^0_{out}, P^0_{in} \cap P^0_{out} = \emptyset$
Expresses launch of a task with a new set of processes.
SUB: $P^0_{in} \to \emptyset$
Expresses termination of a task executed by a set of processes.
SPLIT: $P^0_{in} \to P^k_{out}, \bigcap_i P^i_{out} = \emptyset \wedge \bigcup_i P^i_{out} = P^0_{in}$
Expresses splitting of a task into subtasks.
UNION: $P^i_{in} \to P^0_{out}, \bigcup_i P^i_{in} = P^0_{out},$
Expresses merging subtasks.
GROINK: $P^0_{in} \to \{P^0_{out}, P^1_{out}, P^2_{out}\}, P^1_{out} \subset P^0_{out} \wedge P^2_{out} = P^0_{in} \setminus P^0_{out} \cup P^1_{out}.$
Expresses adding/removing processes to/from a task executed by a PSet.

Principle 3: Process changes are described as set operations

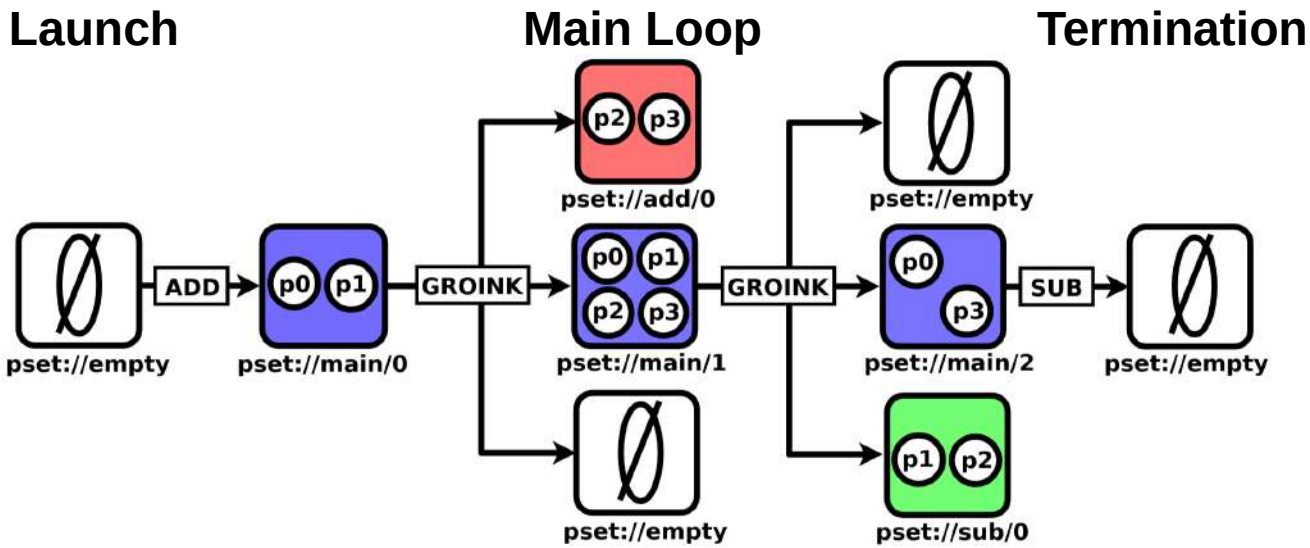
- **Rationale:**
  - Relates process changes to existing PSets
  - Generic abstraction for different senarios



ADD: $P^0_{in} \to P^0_{out}, P^0_{in} \cap P^0_{out} = \emptyset$
Expresses launch of a task with a new set of processes.
SUB: $P^0_{in} \to \emptyset$
Expresses termination of a task executed by a set of processes.
SPLIT: $P^0_{in} \to P^k_{out}, \bigcap_i P^i_{out} = \emptyset \wedge \bigcup_i P^i_{out} = P^0_{in}$
Expresses splitting of a task into subtasks.
UNION: $P^i_{in} \to P^0_{out}, \bigcup_i P^i_{in} = P^0_{out}$,
Expresses merging subtasks.
GROINK: $P^0_{in} \to \{P^0_{out}, P^1_{out}, P^2_{out}\}, P^1_{out} \subset P^0_{out} \wedge P^2_{out} = P^0_{in} \setminus P^0_{out} \cup P^1_{out}$.
Expresses adding/removing processes to/from a task executed by a PSet.

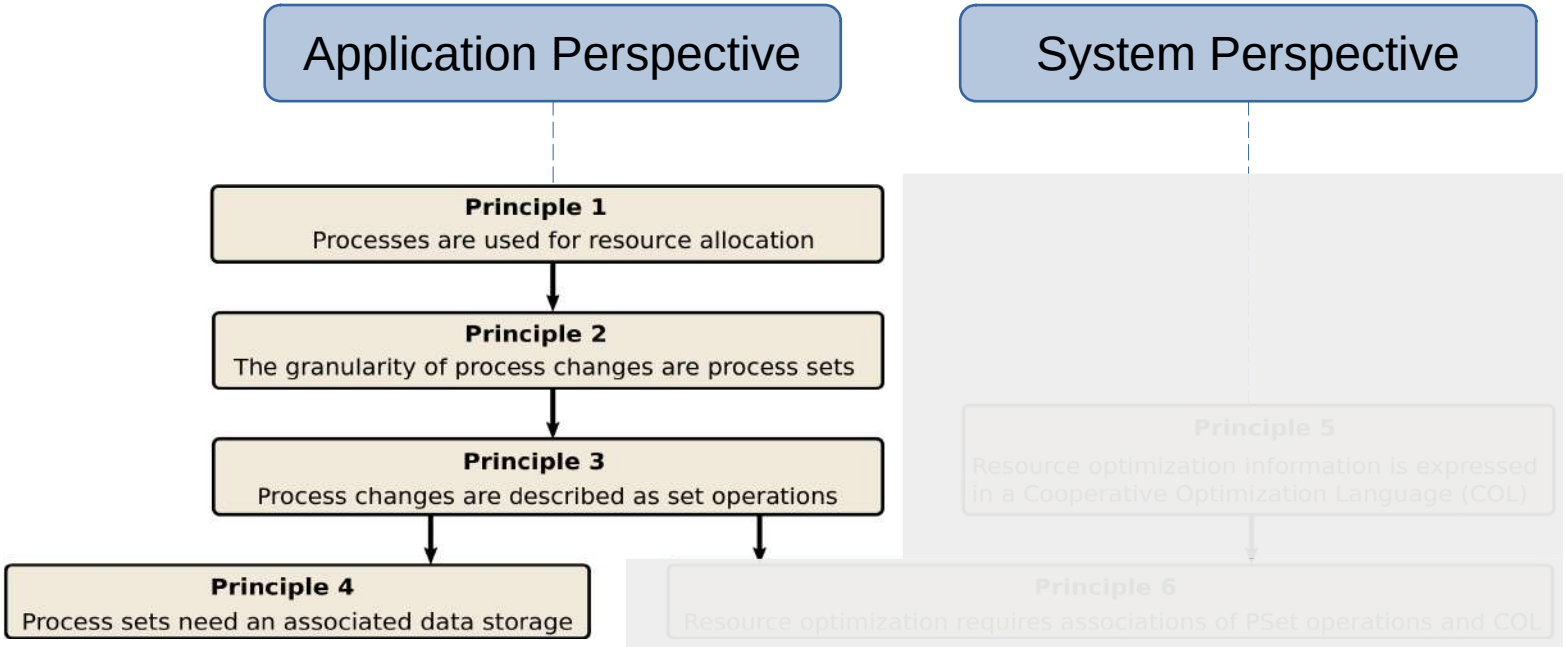
Principle 3: Process changes are described as set operations

- **Example:** Resizing MPI_COMM_WORLD



D. Huber, S. Iserte, M. Schreiber, A. J. Peña and M. Schulz, "Bridging the Gap Between Genericity and Programmability of Dynamic Resources in HPC," ISC High Performance 2025 Research Paper Proceedings (40th International Conference), Hamburg, Germany, 2025, pp. 1-11.

Principle-Driven Design

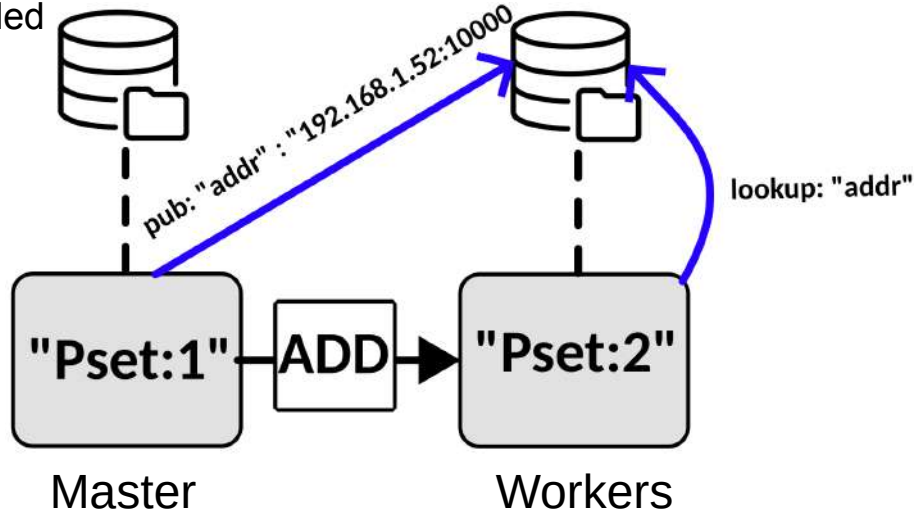


Principle 4: Process Sets need an associated data storage

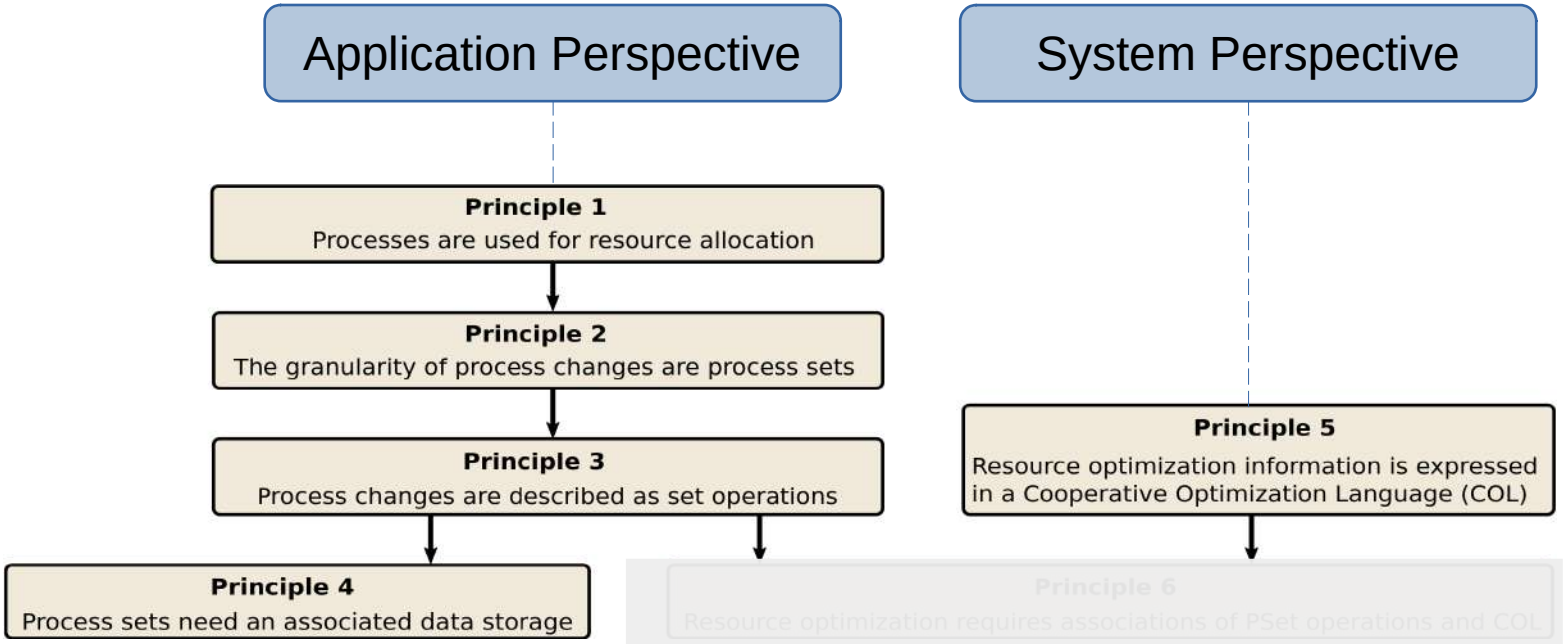
Dynamic Process Environment:

- Possibly no direct communication channels between PSets/processes
- Necessary information not always known a-priori
- Generic way of exchanging „control“ data needed

Example: Master-Worker



Principle-Driven Design



Principle 5: Optimization Information is expressed in a COL

Definition “Cooperative Optimization Language (COL)”: A Cooperative Optimization Language expresses optimization information for DRM in a **cooperative**, **local**, and **language-based** way.

Principle 5: Optimization Information is expressed in a COL

Definition "Cooperative Optimization Language (COL)": A Cooperative Optimization Language expresses optimization information for DRM in a **cooperative**, **local**, and **language-based** way.

Cooperative

- Actors contribute local information toward a global goal
- Global optimization supports local resource needs

Principle 5: Optimization Information is expressed in a COL

Definition "Cooperative Optimization Language (COL)": A Cooperative Optimization Language expresses optimization information for DRM in a **cooperative**, **local**, and **language-based** way.

Cooperative

- Actors contribute local information toward a global goal
- Global optimization supports local resource needs

Local

- Provide optimization information solely for themselves
- No reliance on global system knowledge
- Resource request are „implicit" through provided optimization information

Principle 5: Optimization Information is expressed in a COL

Definition "Cooperative Optimization Language (COL)": A Cooperative Optimization Language expresses optimization information for DRM in a **cooperative**, **local**, and **language-based** way.

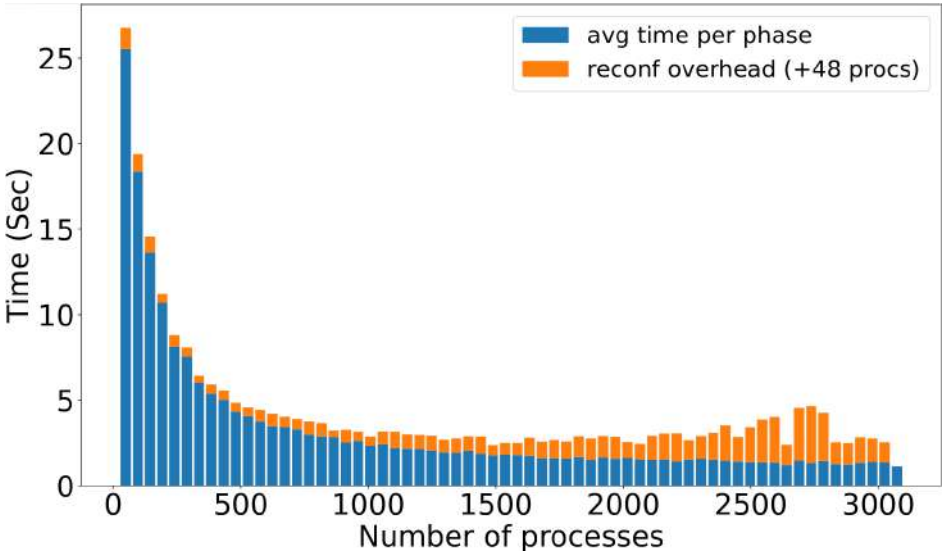
Cooperative
- Actors contribute local information toward a global goal
- Global optimization supports local resource needs

Language-based
- A domain-specific language
- Intermediate Representation

Local
- Provide optimization information solely for themselves
- No reliance on global system knowledge
- Resource request are „implicit" through provided optimization information

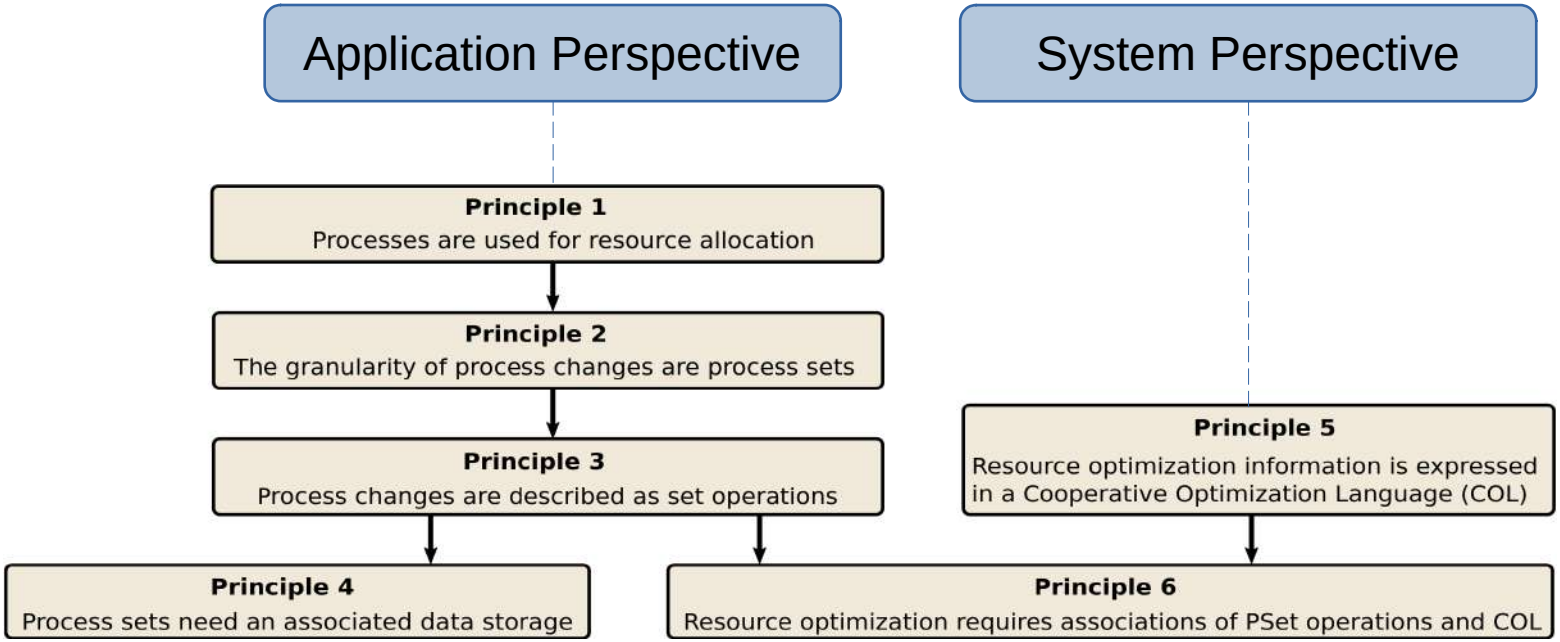
Principle 5: Optimization Information is expressed in a COL

Example: Simple optimization information for iterative workload

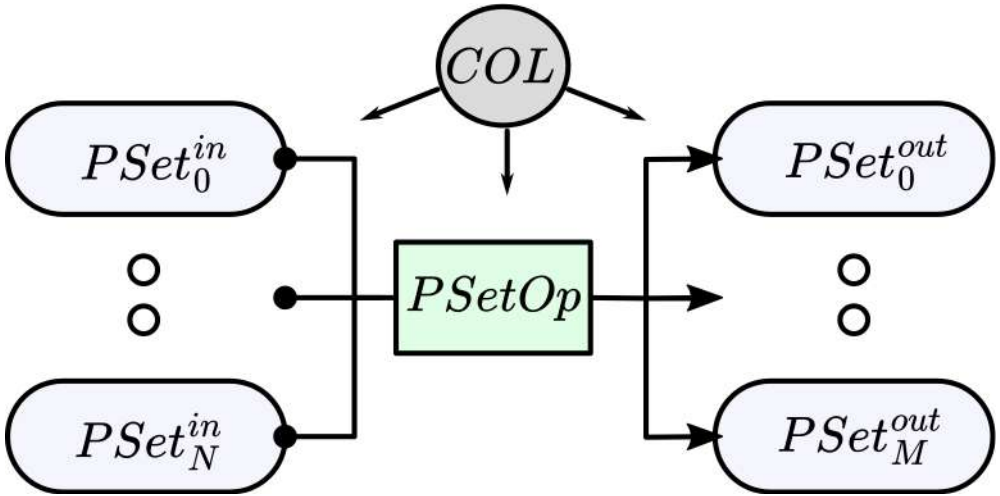



```
{
  opt_info : {
      perf_model : amdahl
      perf_params : t_s=1,t_p=300
      oh_model : 1.5*N + 4
      min_ram_per_proc : 10
      procs_per_node : 48
      min_procs : 96
      max_procs : 9600
    }
}
```

Principle-Driven Design



Principle 6: Association between Pset Operations and COL



OUTLINE

The Need for Common Interfaces

Design Principles

Minimal Interfaces

Prototype & Early Experiences

Minimal Interfaces: PSet Operations

```
//Indicates support for a PSetOp
async (op, in_sets, col_obj) -> out_sets
    op:          The type of the PSetOp
    in_sets:     List of input PSet names
    col_obj:     Info expressed by the COL
    out_sets:    List of output PSet names
```

Listing 3. Minimal interface for indicating support for a PSetOp.

Minimal Interfaces: PSet Dictionary

```
// Publishes key-value pairs
(pset_name, key-vals) -> None
    pset_name:  ID for data storage
    key-vals:   Key-values to publish

// Retrieves key-value pairs
(pset_name, keys) -> key-vals
    pset_name:  ID for data storage
    keys:       Keys to be looked up
    key-vals    Retrieved key-values
```

Listing 4. Minimal interface for accessing the data storage associated with a PSet

OUTLINE

The Need for Common Interfaces

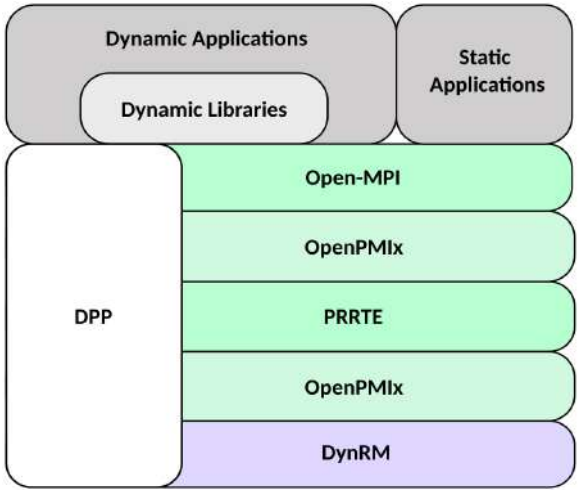
Design Principles

Minimal Interfaces

Prototype & Early Experiences

Prototype and Early Experiences

PetSc [1] Parallel numerical software library	**DynLAIK [13]** Data distribution for dynamicity and fault tolerance	**LibPFASST [3]** Parallel-in-time integration library
P4est [2] Adaptive Mesh Refinement library	**XBraid [4]** Multigrid parallel-in-time integration	**MPDATA3D [12]** 3D semi-Lagrangian multiscale fluid solver
DMR-API [6] Dynamic resources API for computational kernels	**AMT-GLB [11]** Asynchronous Many-Task Runtime System	**DPPInSitu [10]** Library for dynamic, asynchronous in-situ techniques



Summary and Future Work

- **Goal: Common, standardized Dynamic Resource Management Interfaces**
- **Approach:**
  - **Generic Design Principles** for parallel programmin models and middleware
  - **Prototype** and collaborative development environment
- **Future Work:**
  - **Standardization:** COL, MPI, PMIx, …
  - **Integration** with production reosurce managers

Contact
domi.huber@tum.de