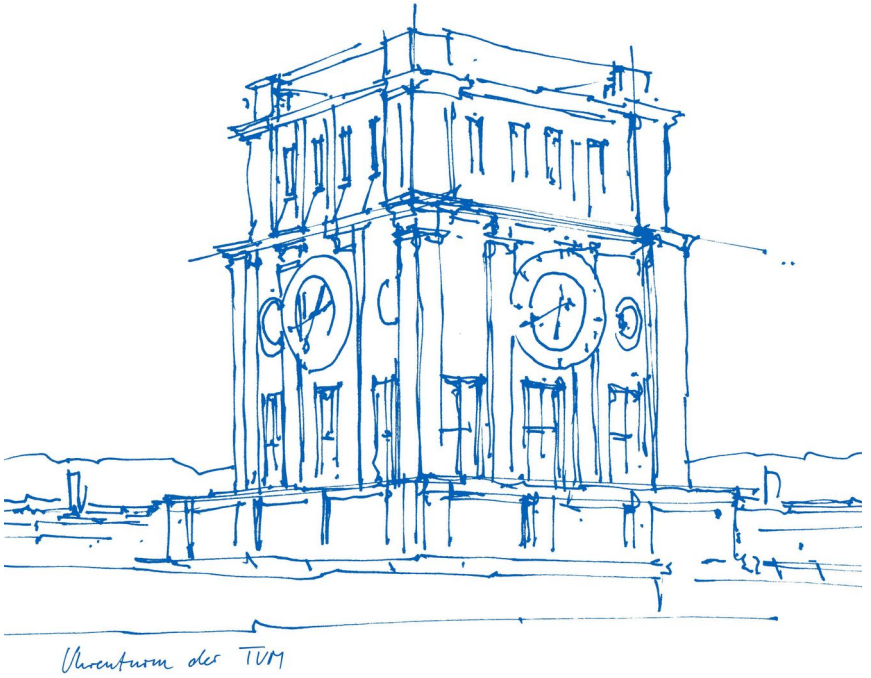
Dynamic Resource Management with MPI Sessions

Dominik Huber, **Martin Schreiber***
Computer Architecture and Parallel Systems
Technical University of Munich

* also at LJK / Université Grenoble Alpes



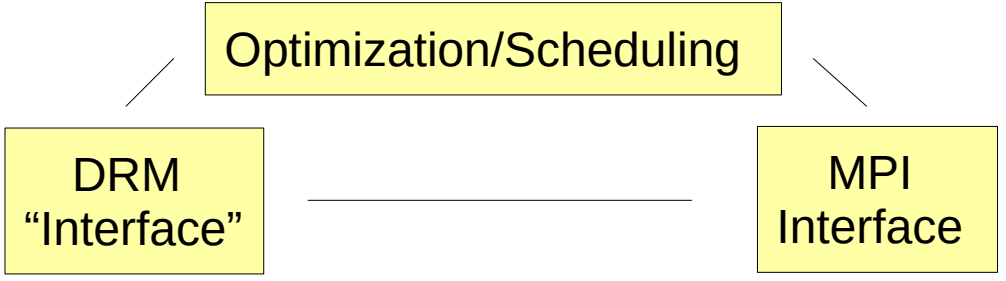
MPI Sessions WG, 1st August 2022

Basic Design Principles

- **Goal:**
  A flexible, general API for dynamic resource management (DRM)

- **Approach:**
  - All kinds of applications need to be supported
  - Resources are **Sets**
  - Resource Changes are **Set Operations**
  - **Graph-based view** on resources

- **Separation of concerns:**



- **Method:**
  Decompose into basic building blocks, then combine for gaining performance

Outline

Basic Components of Resource Changes

Version 1: Decomposition

Version 2: Combination

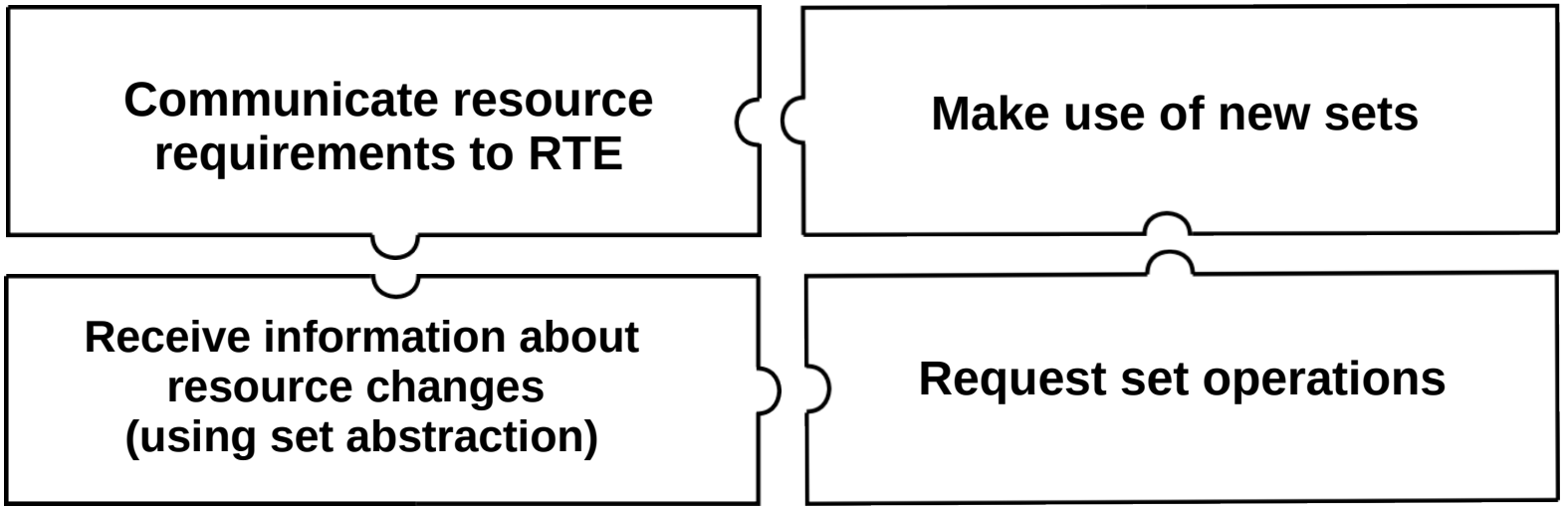
Code Example: Loop-based

Summary

Definition of Sets

- **Sets** are **collections of “resource-related elements”** (CPUs, procs, threads)
- **Sets** are labeled with **unique names** (e.g. URIs)
- **Sets** might **cache attributes**
- **Sets** are **immutable**
- **Sets** are **defined by the RTE / implementation**
- **Sets** might be defined based on **set operation requests from applications**

Components for Dynamic Resource Management



Outline

Basic Components of Resource Changes

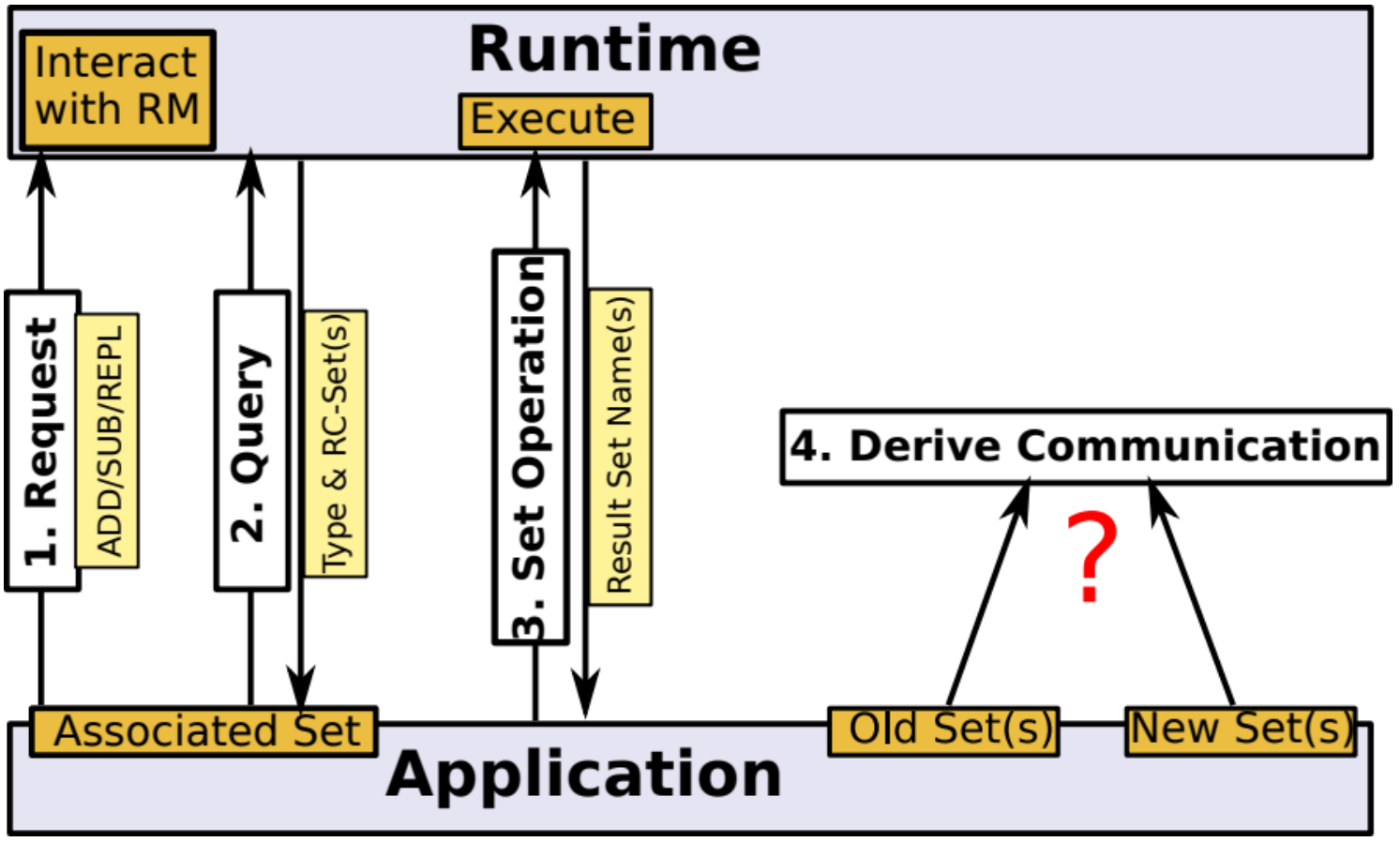
Version 1: Decomposition

Version 2: Combination

Code Example: Loop-based

Summary

Version 1


Runtime
Interact with RM
Execute
1. Request
ADD/SUB/REPL
2. Query
Type & RC-Set(s)
3. Set Operation
Result Set Name(s)
4. Derive Communication
?
Associated Set
Old Set(s)
New Set(s)
Application

Version 1: Possible API (1/2)

1. Request Resource Changes

- `MPI_Session_Dyn_request_res_change(IN: associated_pset, rc_type, rc_params)`

2. Query Resource Changes

- `MPI_Session_Dyn_Test_res_change(IN: associated_pset, OUT: flag)`
- `MPI_Session_Dyn_Recv_res_change(IN: associated_pset, OUT: type, rc-psets)`

3. Pset operations:

- `MPI_Session_Dyn_pset_create_op (IN: op_type, op_params, OUT: result_psets)`

Version 1: Possible API (2/2)

4. Derive Communication from new Pset(s)

- `MPI_Session_set_Pset_info(IN: pset, IN: info)`
- `MPI_Session_bcast_Pset_info(IN: pset, INOUT: info)`
- `MPI_Session_Test_Exists(IN: pset, OUT: flag)`

Global Dictionary?!

MPI-style communication?!

Further routines:

- `MPI_Session_get_default_pset(IN: session, OUT: default_pset)`
- `MPI_Session_get_primary_process(IN: pset, OUT: flag)`
- `MPI_Session_Pset_IsProcessIncluded(IN: pset, OUT: flag)`
- `MPI_Session_Pset_move/delete(IN: pset)`

Outline

Basic Components of Resource Changes

Version 1: Decomposition

Version 2: Combination

Code Example: Loop-based

Summary

Definition: Resource Change

→ „*A Resource Change is a **directed acyclic graph of sets & set operations***“

Definition: Set Operation

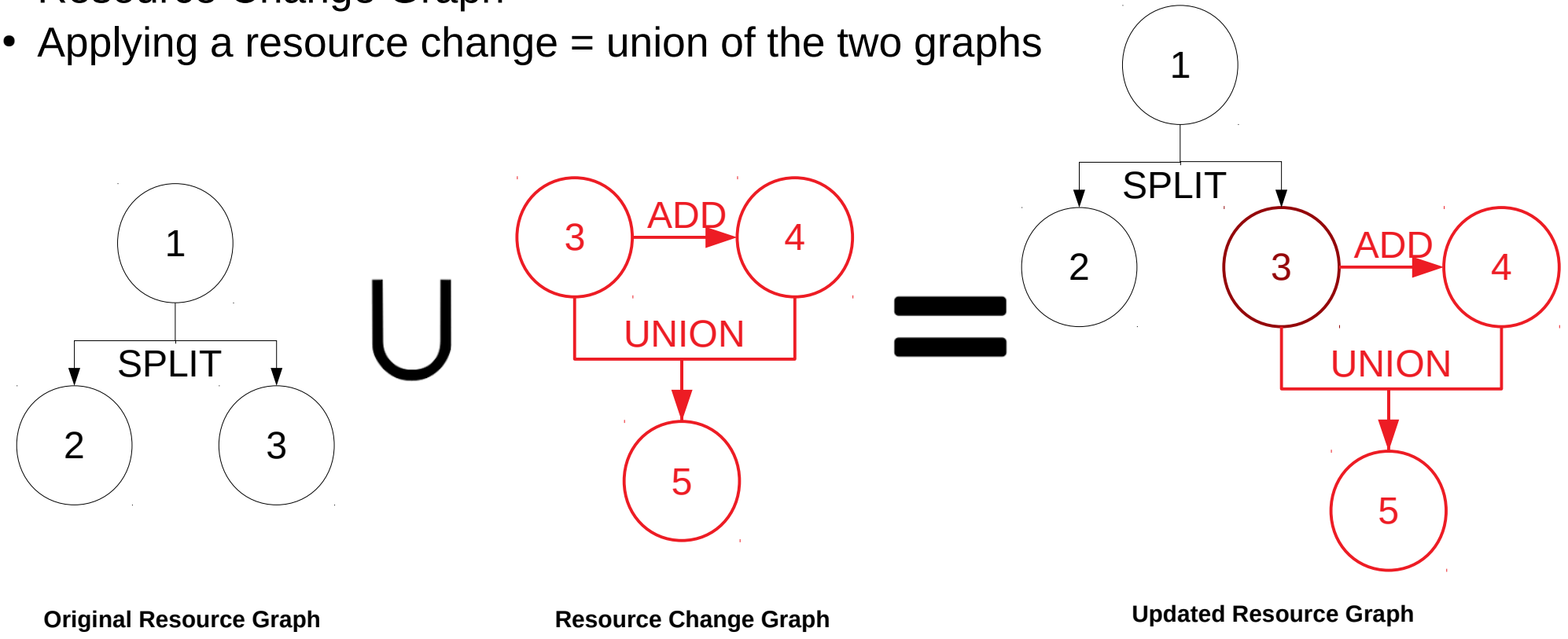
„*A set operation creates one or more new sets (possibly with associated attributes)*“

- **Set Operation Types (Question to audience: Anything missing?):**
  - ADD: creates a set including the newly allocated resources
  - SUB: creates a set including the resources to be removed
  - REPLACE: creates 2 sets including the new resources / resources to be removed
  - UNION: creates a set which is the *union* of other sets
  - DIFFERENCE: creates a set which is the *difference* of other sets
  - INTERSECTION: creates a set which is the *intersection* of other sets
  - SPLIT: Creates non-overlapping subsets of a set, having the specified cardinalities
- **Attributes:** Arbitrary, application-defined + possibly some default attributes

Graph-based view on resources

- App/Job Resource Graph
- Resource Change Graph
- Applying a resource change = union of the two graphs

DAG (nodes = sets & hyper-edges = set operations)

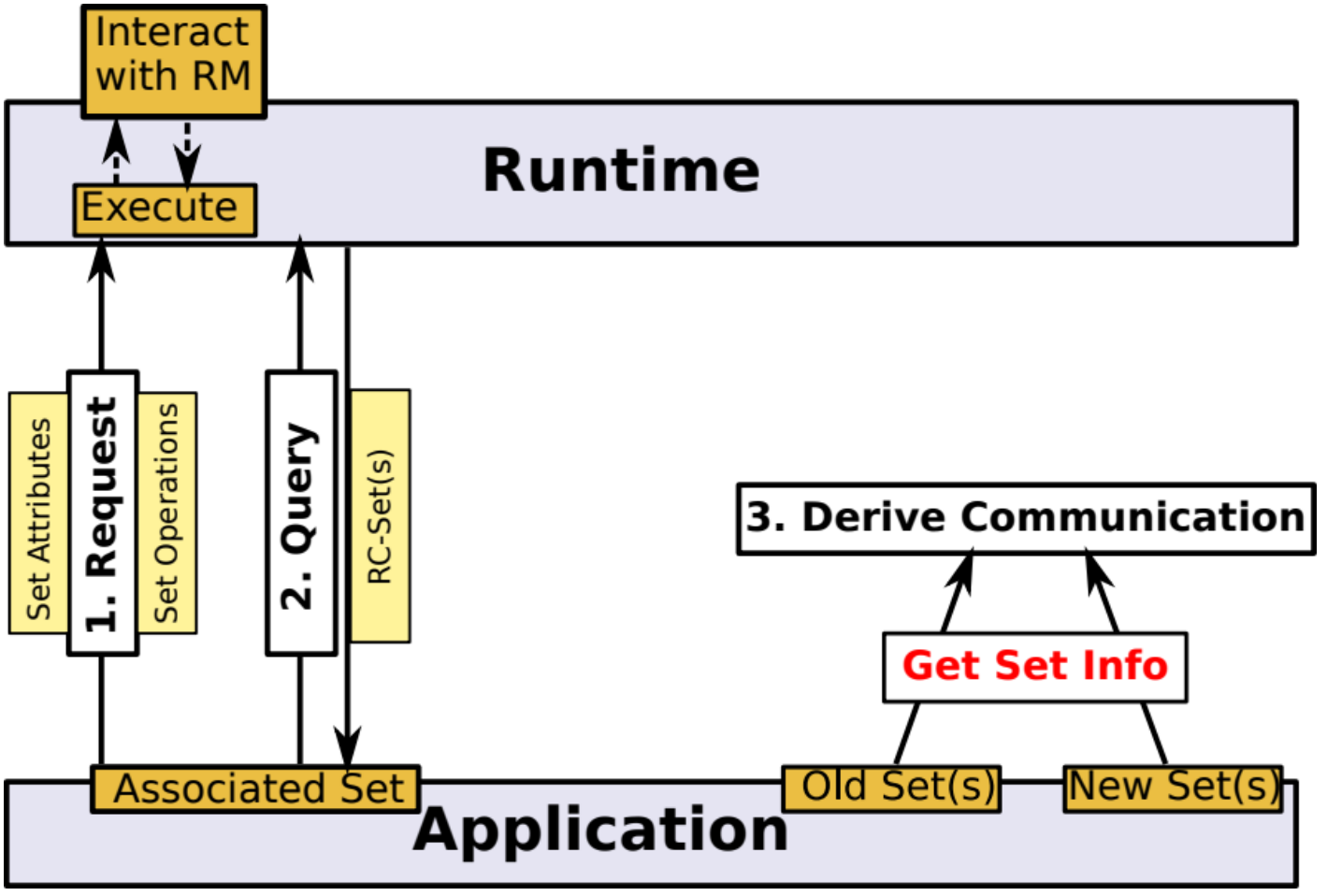


Original Resource Graph

Resource Change Graph

Updated Resource Graph

Version 2


Interact with RM
Runtime
Execute
Set Attributes
1. Request
Set Operations
2. Query
RC-Set(s)
3. Derive Communication
Get Set Info
Associated Set
Old Set(s)
New Set(s)
Application

Version 2: Possible API

1. Request Resource changes

- `MPI_Session_Dyn_create_pset_op_handle(IN: op_type, op_parameters, OUT: op_handle)`
- `MPI_Session_Dyn_request_res_change(IN: assoc_pset, [op_handles], pset_attributes)`

2. Query Resource Changes

- `MPI_Session_Dyn_Test_res_change(IN: associated_pset, collective_pset, OUT: flag)`
- `MPI_Session_Dyn_Recv_res_change(IN: assoc_pset, coll_pset, OUT: new_psets, n_psets)`

3. Derive Communication

- `MPI_Session_Test_Exists(IN: pset, OUT: flag)`
- `MPI_Session_Pset_get_info (Existing API)`

Instead adjust MPI_Comm_create_from_group

Local Dictionary Lookup!

Further Routines

- `MPI_Session_get_default_pset(IN: session, OUT: pset)`
- `MPI_Session_Pset_IsProcessIncluded(IN: pset, OUT: flag)`
- `MPI_Session_get_primary_process(IN: pset, OUT: flag)`

Could be Session / PSet info

Version 2: Possible API

1. Request Resource changes

- `MPI_Session_Dyn_create_pset_op_handle(IN: op_type, op_parameters, OUT: op_handle)`
- `MPI_Session_Dyn_request_res_change(IN: assoc_pset, [op_handles], pset_attributes)`

2. Query Resource Changes

- `MPI_Session_Dyn_Test_res_change(IN: associated_pset, collective_pset, OUT: flag)`
- `MPI_Session_Dyn_Recv_res_change(IN: assoc_pset, coll_pset, OUT: new_psets, n_psets)`

3. Derive Communication (Existing API)

- `MPI_Session_get_info`
- `MPI_Session_Pset_get_info`

Outline

Loop-based Example (1/4): Initialization


```
MPI_Session_init(MPI_INFO_NULL, MPI_ERROR_RETURNS, &session);

// get the session info
MPI_Session_get_info(session, &info);

// get the default pset, i.e. the PSet containing the processes of the same "launch"
MPI_Info_get(info, MPI_SESSION_PSET_DEFAULT, MPI_PSET_MAX_NAME_LEN, default_pset, &flag);

// Get the "MAIN_PSET" Attribute cached on the default PSet
MPI_Session_Get_Pset_info(session, default_pset, &default_pset_info);
MPI_Info_get(default_pset_info, "MAIN_PSET", MPI_PSET_MAX_NAME_LEN, main_pset, &flag);

if (!flag){
     strcpy(main_pset, default_pset);
}

/* In this example, the 'primary process' of the main_pset requests resource changes*/
MPI_Session_Get_Pset_info(session, main_pset, &main_pset_info);
MPI_Info_get(main_pset_info, MPI_PSET_IS_PRIMARY_PROC, &primary_proc_flag, &flag);

// Pset --> communicator
MPI_Session_group_from_pset(main_pset, &group);
MPI_Comm_create_from_group(group, &comm);
```

Loop-based Example (2/4): Main Loop Structure

```
/* MAIN LOOP */
while(1 /* work */){

     /*
      * Application dependent load:
      * Possible a balancing and work step
      */
     rebalance_step(/* */);
     /*...;*/
     work_step();


     /* REQUEST RESOURCE CHANGE */
     if (primary_process_flag && /* need to request a resource change */){

          /* Send Resource Change Request (see Part 3)*/

     }
     /* QUERY RESOURCE CHANGE */
     if (/* Ready to receive resource change, e.g. every 10th iteration */){

          /* Receive Resource Change Info & process it (see Part 4)*/

     }

}
Finalization:
     MPI_Session_finalize();
```

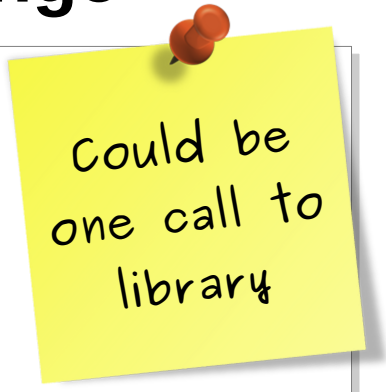
Loop-based Example (3/4): Request Res. Change

```
MPI_rc_op_handle op_handles[2];     // NEW TYPE
MPI_Info add_info, union_info;      // MPI_Info_create omitted for the sake of brevity
MPI_Info new_pset_attributes[2];    // MPI_Info_create omitted for the sake of brevity

/***********************************
 * create the ADD operation handle *
 ***********************************/
MPI_Info_set(add_info, MPI_NUM_PROCS, "8");                     // Set info for 'ADD' (Here: Request 8 new procs)
MPI_Info_set(add_info, MPI_PSET_LABEL, "label://add_pset");     // Label for the 'ADD-PSet'
// NEW API: Create a resource change operation handle
MPI_Session_create_rc_op_handle(
                                MPI_RC_ADD,             // IN: The operation to create
                                add_info,               // IN: Info object containing parameters of the operation
                                &op_handles[0]          // OUT: A MPI_rc_op_handle (Used in MPI_Session_request_res_change)
                            );

/***********************************
 * create the UNION operation handle *
 ***********************************/
MPI_Info_set(union_info, MPI_UNION_PSET1, main_pset);
MPI_Info_set(union_info, MPI_UNION_PSET2, "label://add_pset");
MPI_Info_set(union_info, MPI_PSET_LABEL,  "label://union_pset");       // Label for 'UNION-PSet'
MPI_Session_create_rc_op_handle(MPI_RC_UNION, union_info, &op_handles[1]);

/***********************************
 * Send the REQUEST to the RTE     *
 ***********************************/
// Set Attributes for Resource Change PSets
MPI_Info_set(new_pset_attributes[0], "MAIN_PSET", "label://union_pset");     // 'MAIN_PSET' attribute for 'ADD-PSet'
new_pset_attributes[1] = MPI_INFO_NULL;                                      // No Attributes cached on 'UNION-PSet'
// NEW API: Send a resource change request to the RTE
MPI_Session_request_res_change(
                                2,                      // IN: Number of operations to be performed by this request
                                main_pset,              // IN: The name of the PSet, this res change is associated with
                                op_handles,             // IN: The MPI_rc_op_handles describing the operations of the request
                                new_pset_attributes     // IN: The MPI_Info object(s) containing the attributes for the new PSets
                            );
```



Loop-based Example (4/4): Receive Res. Change

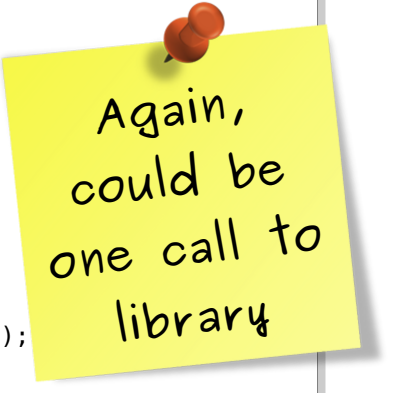
```
/* Receive the PSets created by this res. change and find the one which caches the 'MAIN_PSET' attribute */
// NEW API
MPI_Session_Dyn_Recv_res_change(
                        main_pset,        // IN: The name for which resource change information should be queried
                        main_pset,        // IN: The name of the PSet, across which this query is collective/consistant
                        rc_psets,         // OUT: (char**) array of PSet names, which were created by this res. change
                        &num_psets        // OUT: The number of PSets created by this res. change (0 if no res change)
                        );

if(num_psets > 0){ // If there was a resource change
      /* Find the new main PSet */
      for(int i = 0; i < num_psets && !flag; i++){

            // Check for 'MAIN PSET' attribute
            MPI_Session_Get_Pset_info(session, rc_psets[i], &info);
            MPI_Info_get(info, "MAIN_PSET", MPI_PSET_MAX_NAME_LEN, new_main_pset, &flag);

            if(flag){
                  // Check if this procs is included in new main PSet. If not, it needs to finalize
                  MPI_Session_Get_Pset_info(session, new_main_pset, &main_info);
                  MPI_Info_get(main_info, "MPI_PSET_IS_PROC_INCLUDED", sizeof(incl_flag), &incl_flag, &flag2);
                  if(!incl_flag){
                        goto: Finalization
                  }
                  // Check if this process is the primary process of the new main PSet
                  MPI_Info_get(main_info, "MPI_PSET_IS_PRIMARY_PROC", &primary_proc_flag);
            }
      }

      // Pset --> communicator
      MPI_Session_group_from_pset(new_main_pset, &group);
      MPI_Comm_create_from_group(group, &comm);
      strcpy(main_pset, new_main_pset, ...);
}
```



Outline

Basic Components of Resource Changes

Version 1: Decomposition

Version 2: Combination

Code Example: Loop-based

Summary

Summary

- Goal: A flexible, general API for Dynamic Resource Management
- Set & Graph-based

- Version 1: Decomposition into fine-grained interface
  Question 1: Is there anything missing / not working with MPI/PMIx/SLURM?

- Version 2: Combines/extends functions of Ver. 1 to overcome limitations
  Question 2: Same as before + any showstopper in it?

- Future Work: Find contact points & exchange views with other projects
  - Data Management, Fault tolerance, profiling & load-balancing, system scheduler etc.
    - e.g. **PMIx, SLURM, Time-X**, LAIK, Flex-MPI, DMR-API (BSC)

The Challenge of deriving communication

→ Toy Example: Parameter Sweeps