

The MPI Message-passing Standard

Practical use and implementation (IV)

SPD Course

04/03/2019

Massimo Coppola

COMMUNICATORS AND GROUPS

- Flexible Communication shall provide
 - Safe communication space
 - Scope for communication (esp. collectives)
 - Abstract process naming
 - Option to augment semantics of the communication (by holding “attributes”)
 - With a unified mechanism
- These ideas root in the need to develop interoperable libraries, languages and run-time supports on top of MPI
- Corresponding concepts in MPI
 - Contexts
 - Groups of processes
 - Virtual Topologies
 - Attribute caching
 - Communicators

- Communicators are MPI basic mechanism
- They are global-scope object (created by handshake among processes) made of
 - Groups of processes
 - A group is a local object for naming
 - Context of communication
 - Any information needed to implement communications
 - Attributes : a generic caching mechanism
 - Either user-defined or MPI-implementer defined
 - Virtual Topologies
 - A special mapping of ranks to/from a topology
 - Often implemented via attributes

The General case

- Previous description : **IntraCommunicators**
 - One group of MPI processes with full communication connectivity
- **InterCommunicators** are slightly different
 - Two groups of processes
 - Communication allowed between processes of different groups
 - No virtual topology
- We'll focus on IntraCommunicators

The building bricks

- Group
 - Ordered set of process identifiers
 - From 0 to N-1, consecutive numbering
 - Handles to **Local** Opaque objects:
 - cannot fiddle with it
 - cannot transfer among processes
 - MPI_GROUP_EMPTY special handle for empty
 - MPI_GROUP_NULL invalid handle
- Context
 - Property only defined as associated to communicator
No programming abstraction,
no exhaustive definition in MPI standard
 - Conceptually: separation of communication spaces
 - Pragmatically described as a tag of low-level communications to associate them a communicator
 - Other implementation solutions / more details not provided
- Communicator = Group(s) + Context
 - Note that group is local, context agreement is global

`MPI_GROUP_SIZE(group, size)`

`MPI_GROUP_RANK(group, rank)`

`MPI_GROUP_TRANSLATE_RANKS (group1,
arrSize, ranks1, group2, ranks2)`

- Translate ranks for processes between two groups
- Can receive `MPI_PROC_NULL`
- Can return `MPI_PROC_NULL` for some proc

`MPI_GROUP_COMPARE(group1, group2, result)`

- C prototype

```
int MPI_Group_compare(MPI_Group group1, MPI_Group group2, int  
*result)
```

- Returns `MPI_IDENT`, `MPI_SIMILAR`, `MPI_UNEQUAL`

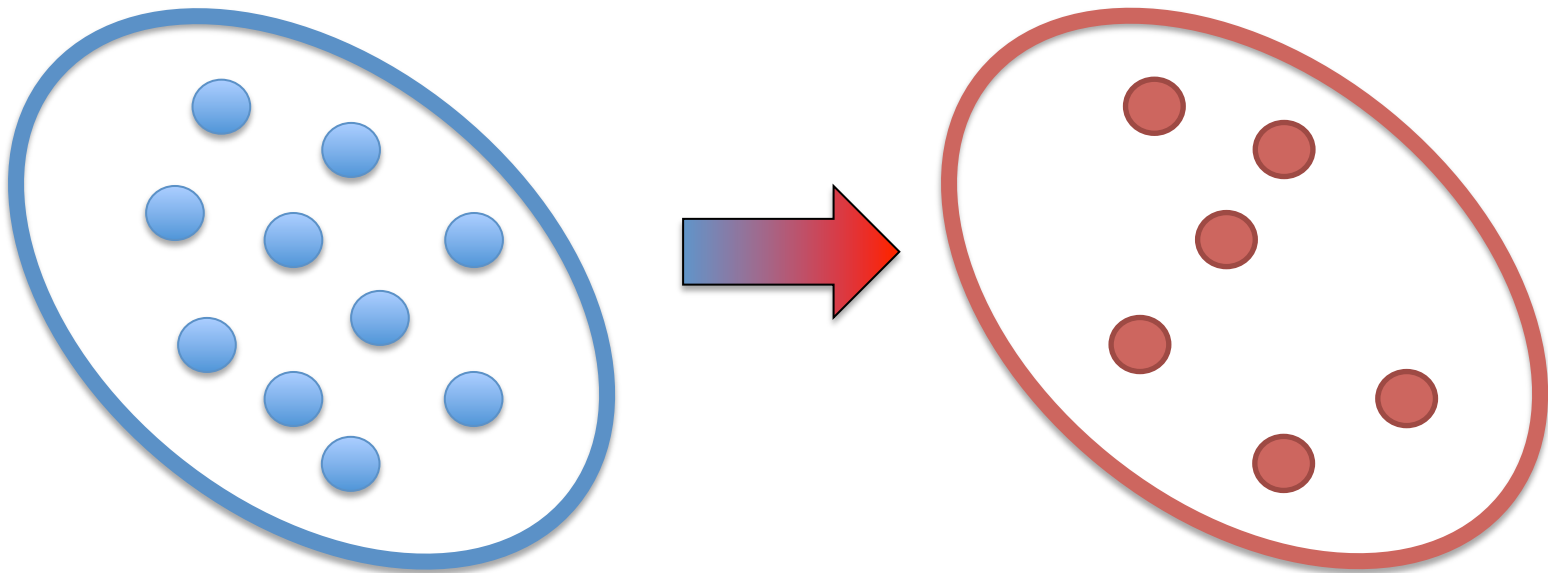
- Groups are local objects → Group operations are cheap
- `MPI_COMM_GROUP(comm, group)`
 - Get group from communicator
- All typical boolean ops:
 - Union, intersection, difference of two groups
 - Order of the first group is prevalent
- `MPI_GROUP_INCL(group, n, ranks, newgroup)`
 - Pick elements from a group, in order, to form a new one
- `MPI_GROUP_EXCL(group, n, ranks, newgroup)`
 - Deletes element from a group
- `MPI_GROUP_RANGE_INCL` ed `EXCL`
 - As above, but define RANGES of ranks
 - Triplets first, last, stride
- `MPI_GROUP_FREE`

- We'll stay with intracommunicators for now
- The cheap ones: get info out of a Comm.
 - `int MPI_Comm_size(MPI_Comm comm, int *size)`
 - `int MPI_Comm_rank(MPI_Comm comm, int *rank)`
 - `int MPI_Comm_compare(MPI_Comm comm1, MPI_Comm comm2, int *result)`
 - `MPI_IDENT` (same Comm) `MPI_CONGRUENT` (same group)
 - `MPI_SIMILAR` (same set of proc.s) `MPI_UNEQUAL`
- The constructors
 - `int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm)`
 - Create a perfect copy (also comm info if info callbacks allow it), but with different context
 - A separate primitive allows replacing the comm. info
- And now for the real thing...

- `int MPI_Comm_create(MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm)`
 - A communicator is always built inside another communicator (Comm_world is the starting point)
 - Cached attributes are lost in newcomm
 - Collective call : all processes in the communicator
 - Should have same parameters from all but...
 - Agreement on group parameter
 - Either all the same (MPI1.1), or all **disjoint** (MPI2.2)
 - May create more comm.s at the same time
 - A process may not be part → returns MPI_NULL_COMM
- `MPI_COMM_FREE()`

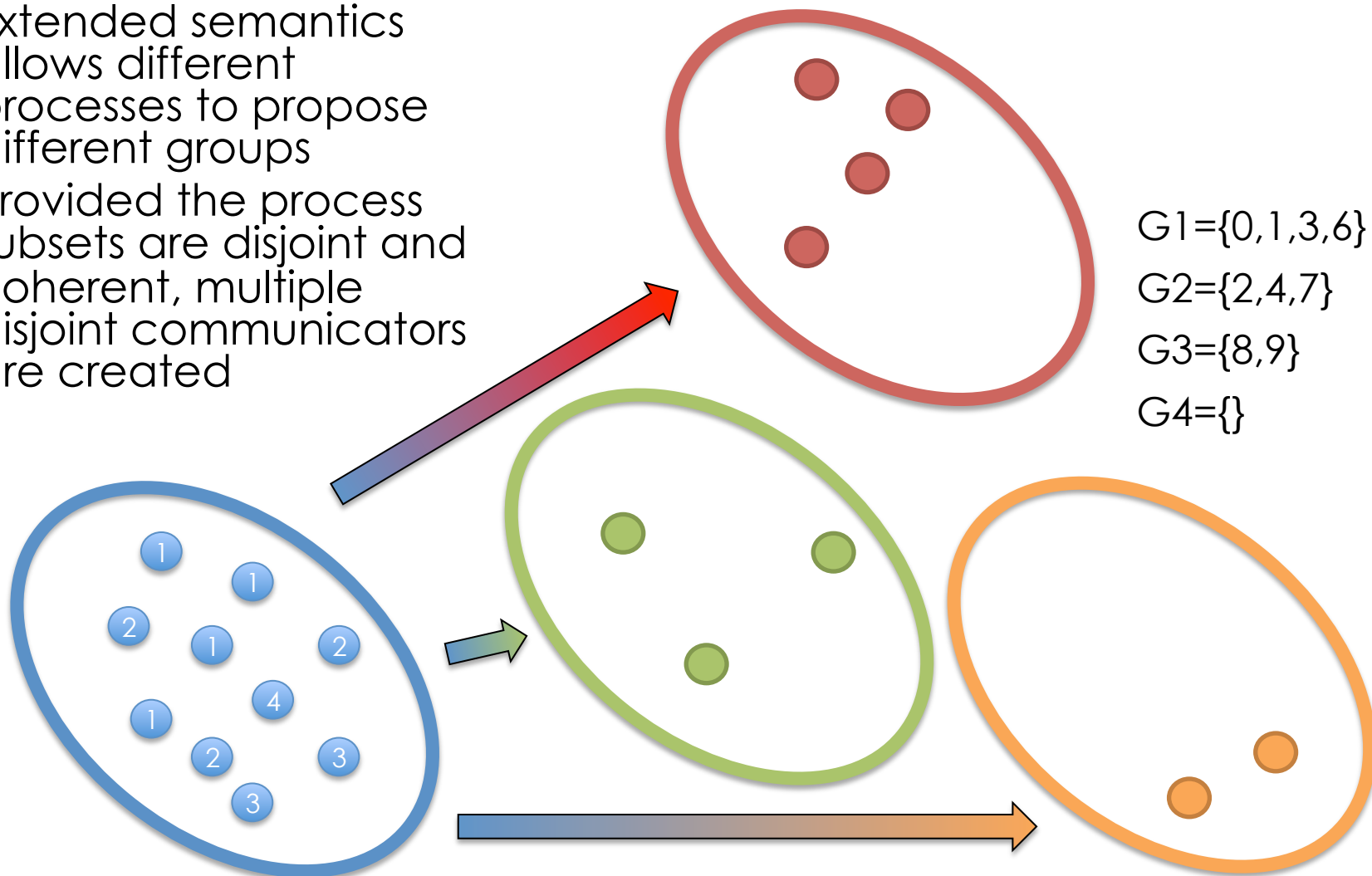
MPI_Comm_create (in MPI 1)

- All processes call with same parameters
 - the same group
- some join the new communicator, some don't (they get MPI_NULL_COMM back)



MPI_Comm_create (in MPI 2)

- Extended semantics allows different processes to propose different groups
- Provided the process subsets are disjoint and coherent, multiple disjoint communicators are created



Communicator Splitting

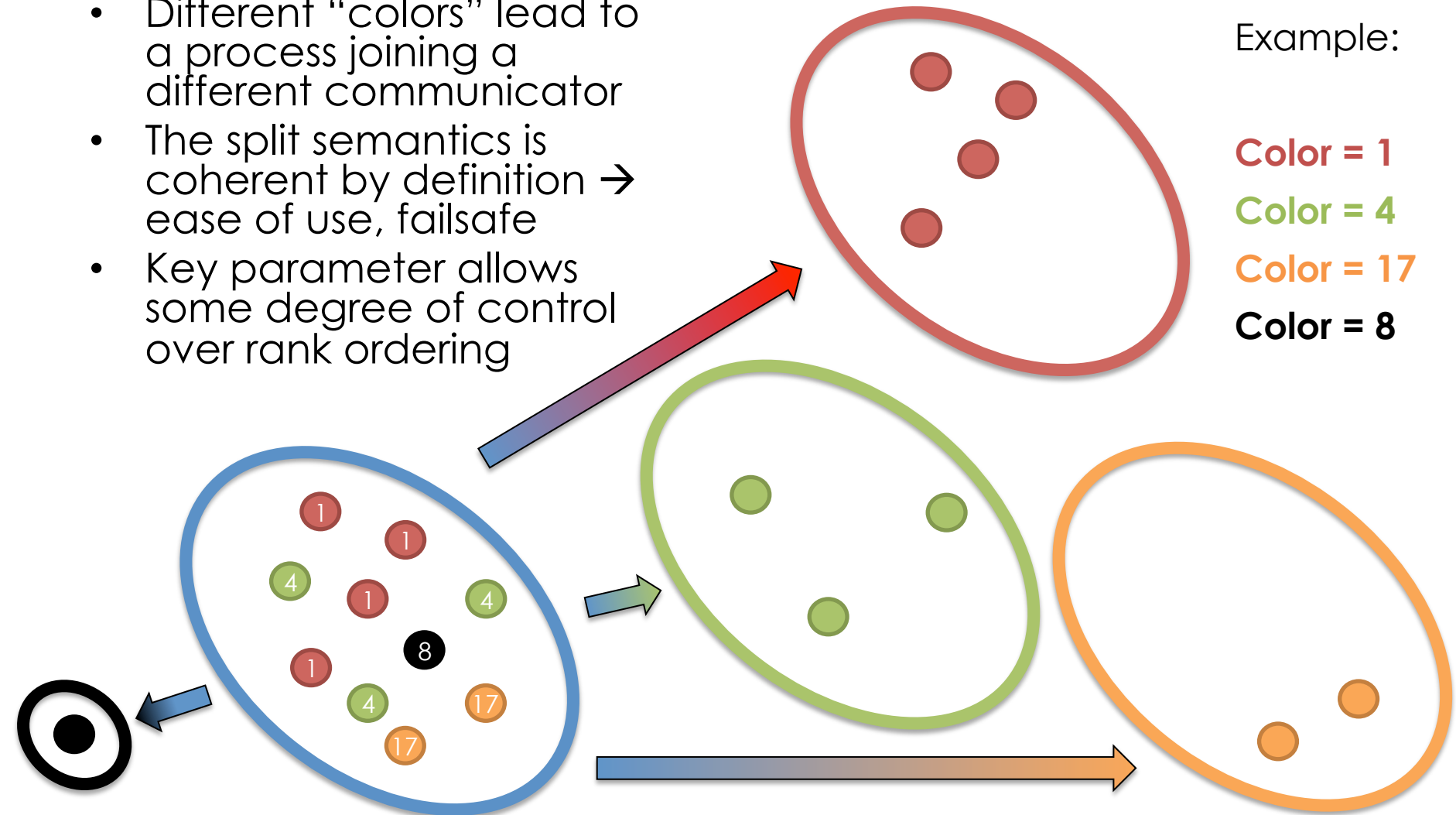
- `int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newcomm)`
 - Collective call
 - *color* and *key* parameters vary among processes
 - $color \geq 0$, or `MPI_UNDEFINED`
 - Describe the splitting of a communicator in order to form several non-overlapping new ones
 - Processes can join the new communicator of the given “color” without knowing its composition in advance
 - a little bit more communication is needed under the hood
 - The *key* parameters allows some control on the ordering of processes (rank assignment) in the new communicator(s)
 - New ranks are ordered by keys, ties broken by old rank

MPI_Comm_split ()

- Different “colors” lead to a process joining a different communicator
- The split semantics is coherent by definition → ease of use, failsafe
- Key parameter allows some degree of control over rank ordering

Example:

- Color = 1
- Color = 4
- Color = 17
- Color = 8



- MPI standard Relevant Material for 4th lesson
 - Chapter 6: up to 6.5 (skip intercommunicators)