

MPI Issues in Parallel Components Environments

Wael R. Elwasif
ORNL

Where to put MPI??

- Framework:
 - Current practice in Ccaffeine.
 - Components **MUST NOT** link against static MPI lib.
 - Is it enough for all apps (mainly MCMD)??
 - What about access from F90 and Python components??

Where to put MPI?? (2)

- Nowhere
 - Each component links against MPI lib.
 - Problems if we only have static libs available.
 - Must agree on communicators, MPI_Init(), ..etc.
 - Not a good solution (actually a BAD one).
- Standalone Port/Component
 - Solves all linking problems.
 - What about legacy codes???

So, what do we need?

- We do need an MPI port (or at least a mini MPI port).
- Components should continue to use MPI directly when possible (we don't want to mess with legacy codes).
- Components need to share a single MPI state.
- We need to have a strategy that would work with static and dynamic MPI libraries.

Proposed Solution – Linux, Mpich

- Shared libraries available:
 - Framework & Components link against mpich libraries (will get single state due to shared-ness)
- Only static libraries:
 - Framework links against “core” libmpich.a
 - Components link against wrapper libraries only (if needed).
- MPI port can work in both cases.
- Portability ... Portability ... Portability ??????

What do we need in an MPI port?

- We can have it all, although a subset for communication space management would suffice for most purposes (remember legacy codes).
- MPISetup port:
 - To be used mainly to handle MCMD situations.
 - Needed to allow manipulation from driver components in any language.

The CCAMPI Component (1)

```
package CCAMPI version 1.0{

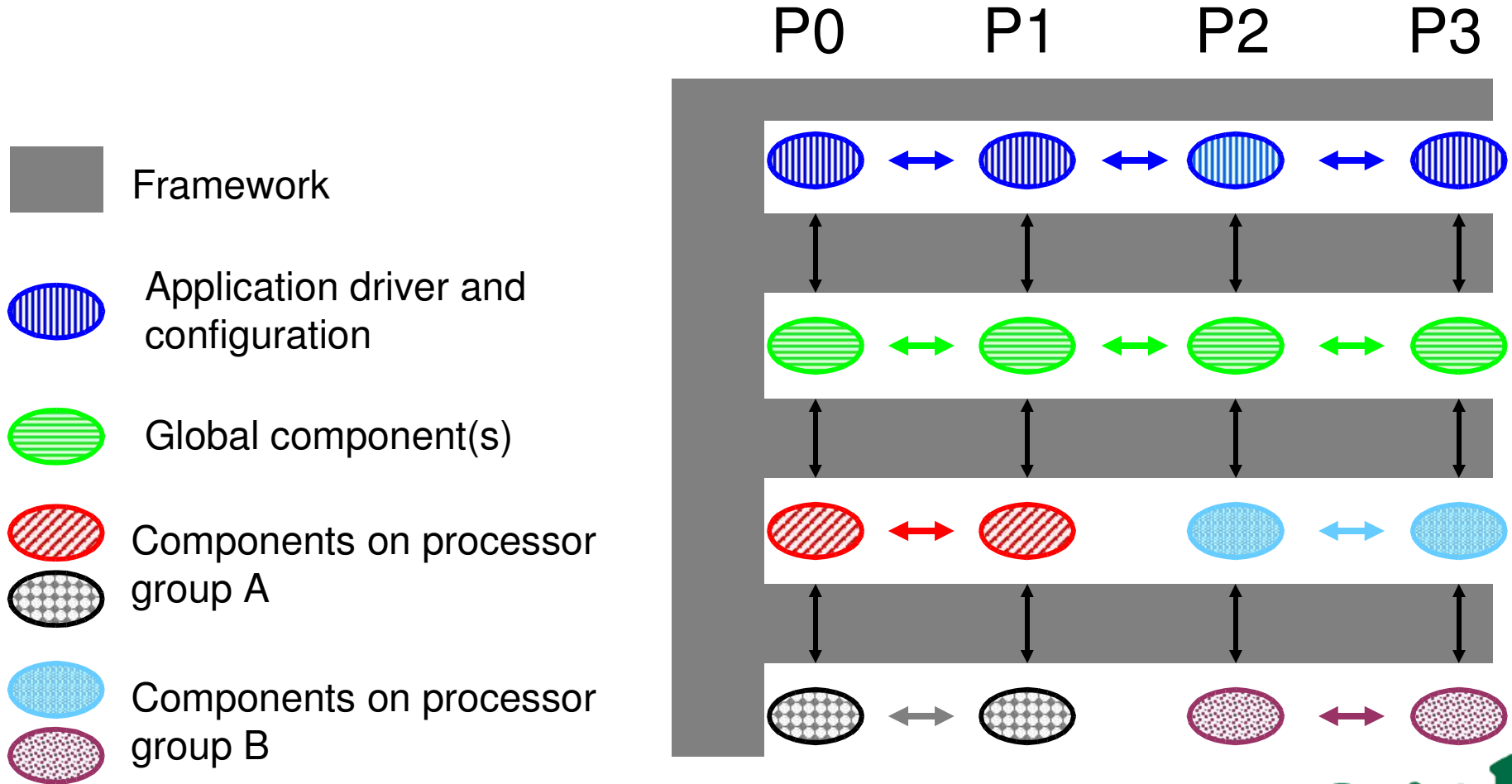
    interface Comm {
        int  getSize(inout int size);
        int  getRank(inout int rank);
        int  getGroup(in Comm oldCom, out Group oldGroup);
        int  create(in Group oldGroup, out Comm newCom);
        int  split(in int color, in int key, out Comm newCom);
        int  free();
        opaque getInternalRep();
        int  setInternalRep(in opaque internalCommRep);
    };
    interface Group {
        int  getSize(inout int size);
        int  getRank(inout int rank);
        int  incl(in int n, in array<int, 1> ranks, out Group ewgroup);
        int  free();
        opaque getInternalRep();
    }
}
```

The CCAMPI Component (2)

```
interface MPISetup extends gov.cca.Port {
    Comm getComWorld();
    int getWorldRank();
    Comm buildCommObj(in opaque internalCommRep);
    Group buildGroupObj(in opaque internalGroupRep);
}
}
package CCAMPIImpl version 1.0 {
    class CommClass implements-all CCAMPI.Comm {
        static CCAMPI.Comm buildCommObj(in opaque internalCommRep);
        int setInternalRep(in opaque internalCommRep);
    }
    class GroupClass implements-all CCAMPI.Group {
        static CCAMPI.Group buildGroupObj(in opaque internalGroupRep);

        int setInternalRep(in opaque internalGroupRep);
    }
    class MPIComponent implements-all CCAMPI.MPISetup,
        gov.cca.Component {}
}
```

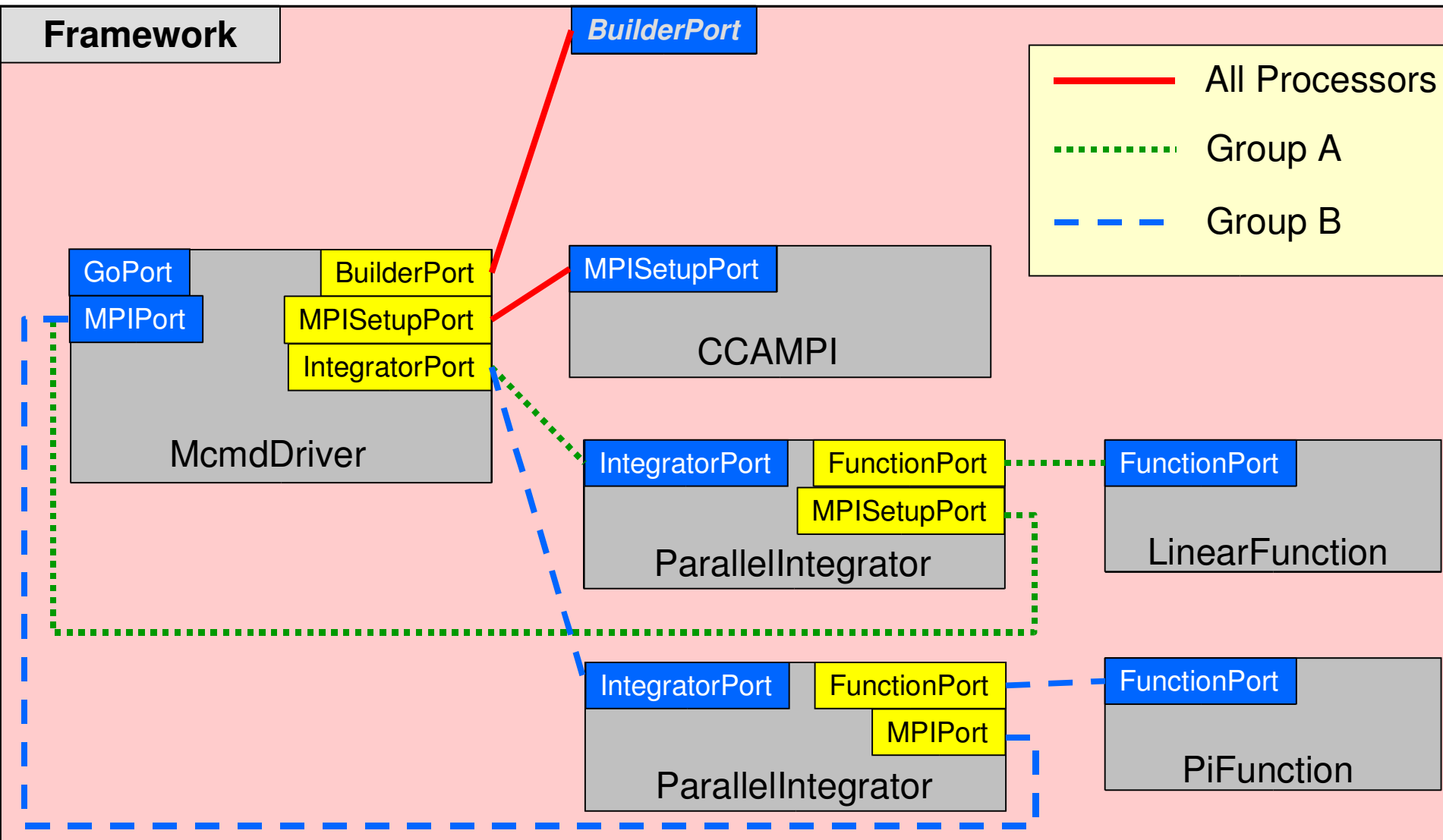
MCMD Within A Single CCA Framework



MCMD using Ccaffeine and MPISetup

- Need to:
 - Load different components on different processors.
 - Allow the driver to make the decision based on “global” application state (including MPI state).
- Solution:
 - Use ***BuilderServices*** to control component loading.
 - Use “mini MPI” component to allow access to MPI “setup” calls from various languages (including Python).
 - Structure components to work on a subset of processors (No MPI_COMM_WORLD)

MCMD Port Connections



The McmdDriver Component

```
package tutorial version 1.0 {  
  
    interface MPIPort extends gov.cca.Port {  
        opaque getMpiComm();      // Return the COMM used by the driver  
        opaque getNewMpiComm();  // Return a new COMM that spans the  
                                 // same set of processes  
    }  
  
    class McmdDriver implements-all gov.cca.ports.GoPort,  
                                     gov.cca.Component,  
                                     MPIPort  
  
    {}  
}
```

MCMD Application Logic

- MCMD Driver:
 - Provides ***MPIPort***
 - Instantiate and connect infrastructure components (e. g. ***MPISetupComponent***).
 - Partition MPI_COMM_WORLD based on application logic.
 - Instantiate and connect other components.
- Parallel components
 - Use ***MPIPort*** port to acquire proper communicator (as determined by the driver).
 - Can use **MPI_COMM_WORLD** if no such connection exists.

MCMD Issues

- *MPIPort* port provided by the driver can export more application-specific methods.
- Alternate simple solution: Each component exports *setCommunicator()*.

Code layout to make life easy(ier)

- Separate Port definition and associated interfaces into a separate package.
- Component class(es) are separated into a separate package.
- `--generate-subdirs`, `--exclude-external`, and `-hide-glue` (new in Babel 0.8.8).
- Scripts to automatically generate makefiles (work in progress)
- Example using MPISetup port follows.

```
babel --server=C++ --repository-path=repository
      -output-directory=mpisetup-comp --exclude-external
      --generate-subdirs -language-subdir --hide-glue CCAMPIImpl
```

```
babel -client=C++ --repository-path=repository
      -output-directory=mpisetup-port --exclude-external
      --generate-subdirs -language-subdir --hide-glue CCAMPI
```