CoreGRID

*Managed by*

European Research Consortium for Informatics and Mathematics
ERCIM

# Combining Data Sharing with the Master-Worker Paradigm in the Common Component Architecture

**Gabriel Antoniu, Hinde Lilia Bouziane, Mathieu Jan, Christian Pérez, Thierry Priol**

**PARIS research-team**
IRISA / INRIA
**Rennes - France**

PARIS IRISA INRIA

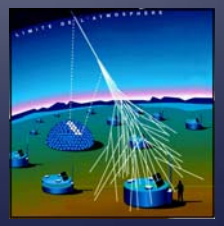
Information Society Technologies

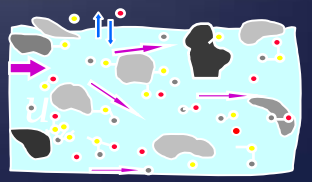
# Outline of the talk

- **Introduction**
  - Context
  - Overview of the Common Component Architecture
- **Data sharing paradigm**
  - Objectives
  - Our proposition : data port model
  - Data sharing on operation invocations
- **Support of the master-worker paradigm**
  - Limits with existing component models
  - A proposed model
- **Data sharing in a master-worker paradigm**
- **Conclusions and perspectives**

# Introduction

- **Context**
  - Complex applications
    - e-Science
  - Different resource kinds
    - multi-core processors
    - SMP machines
    - clusters
    - grids

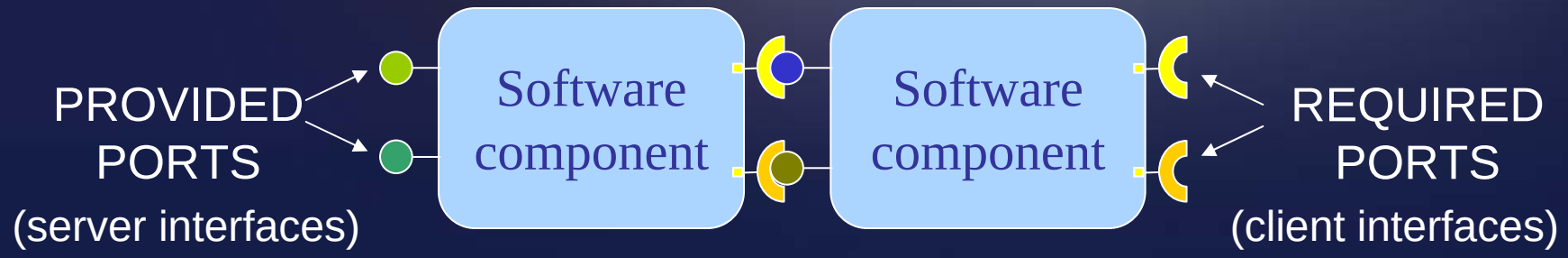





- **Challenges**
  - Simplifying application programming
  - Independence from resource kinds
  - HPC

**Component model approach**

# Software component models



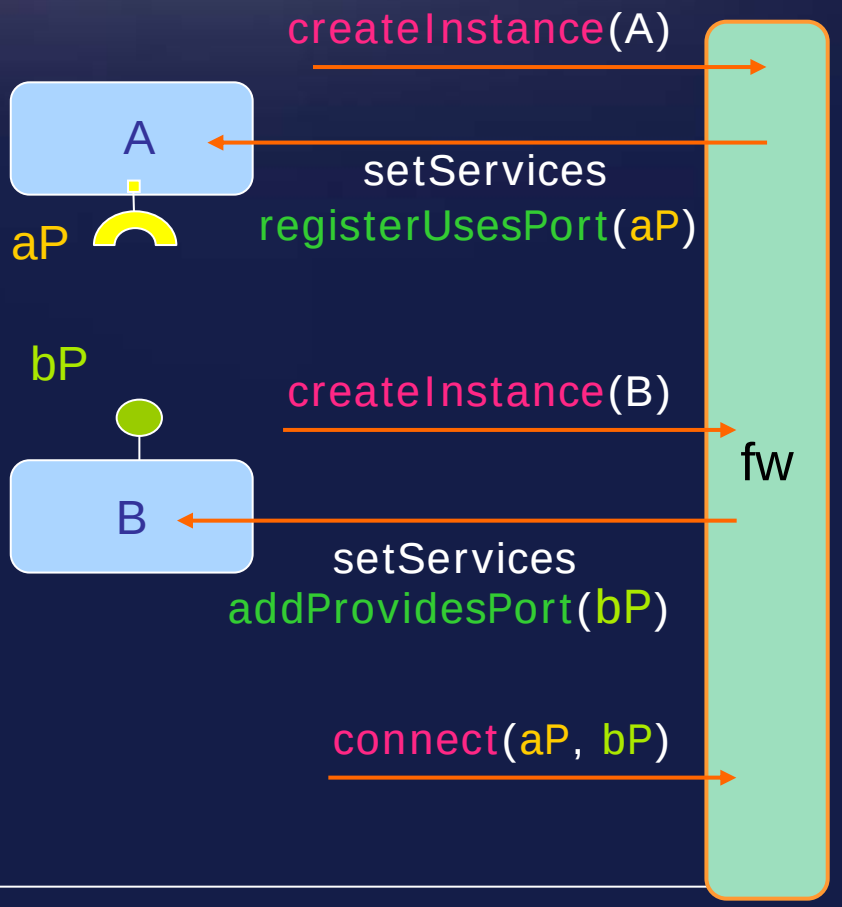

- **Black box**
- **Ports**
  - Method invocations, events/messages/streams
- **Several component models**
  - Common Component Architecture/CCA Forum (CCA)
  - CORBA Component Model/OMG (CCM)
  - Fractal/ObjectWeb
  - Grid.it-ASSIST/UNIPI
  - Etc.

# Overview of the CCA (1/2)


- **CCA forum**
  - US laboratories and academic institutions
- **Specification based on the Scientific Interface Definition Language**
  - Standard interfaces
- **Several implementations**
  - Local frameworks
    - Ccaffeine, SCIRun2, etc.
  - Distributed frameworks
    - Xcat, SCIRun2, Legion-CCA, etc.

# Overview of the CCA (2/2)

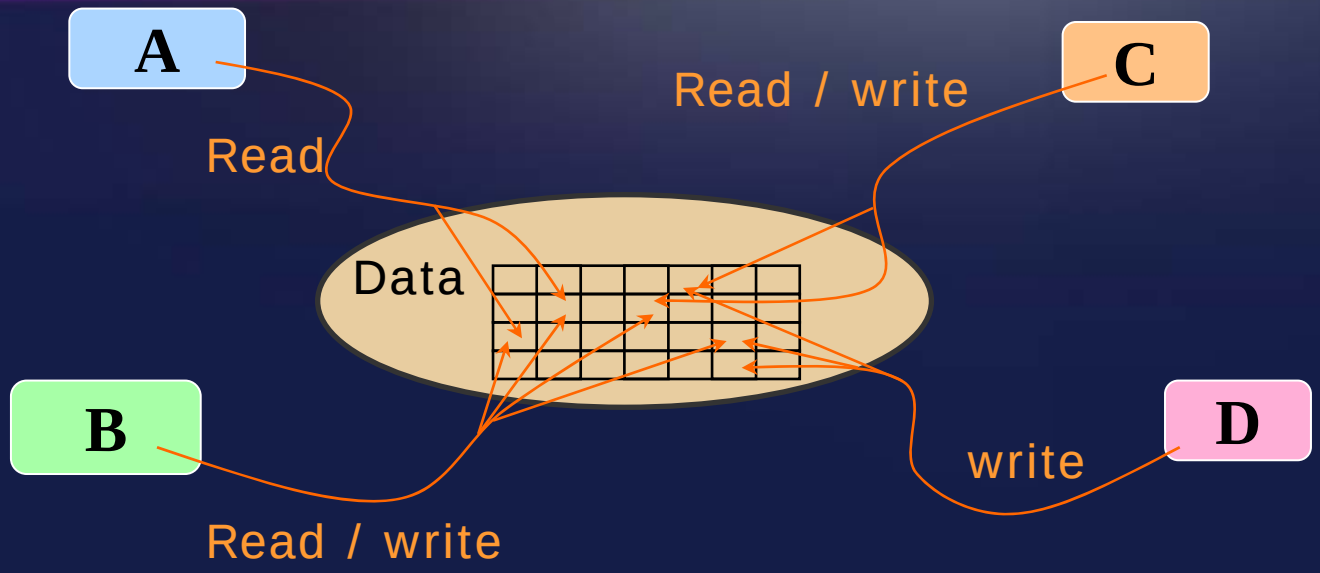
- **All is done at runtime**
- **Examples of specified SIDL interfaces**
  - BuilderService
    - Component creation
      - *createInstance(…)*
    - Composition
      - *connect(…)*
  - Services
    - Port declarations
      - *addProvidesPort(…)*
      - *registerUsesPort(…)*
    - Getting a port reference
      - *getPort(…)*



# Outline of the talk

- **Introduction**
  - Context
  - Overview of the Common Component Architecture
- **Data sharing paradigm**
  - Objectives
  - Our proposition : data port model
  - Data sharing on operation invocations
- **Support of the master-worker paradigm**
  - Limits with existing component models
  - A proposed model
- **Data sharing in a master-worker paradigm**
- **Conclusions and perspectives**

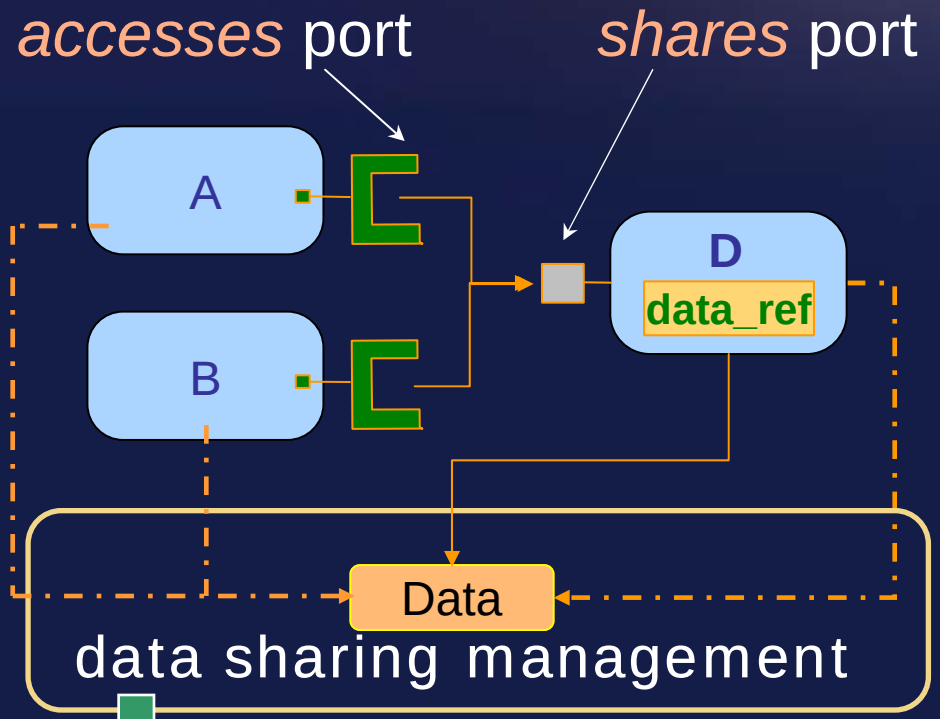
# Problem overview




- **Multiple concurrent accesses to a piece of data**
- **Data localization and management**
  - Intra-process : local shared memory
  - Intra-cluster : distributed shared memory (DSM)
  - Intra-grid : grid data sharing service (JuxMem, PARIS/INRIA )

# Our proposal: data port model

## Principle: transparent data sharing




```
component D {
  shares array<float> u;
};

component A {
  accesses array<float> v;
};
...
```

**Transparent data localization and management**

- Local shared memory, DSM, JuxMem

# Data ports in CCA

## Component implementation example

```
class CompImpl {
  Services srv;
  AccessPort myPort;

  void setServices (Services services ){
    srv = services;
    srv.createAccessPort ("myPort", "arrayFloat",..);
}
  void computeSum(){
    myPort = srv.getPort(“myPort");
    myPort.acquireR();
    ptr = myPort.getPointer();
    size = myPort.get_size();
    for ( i = 0; i < size; i++)
       sum += ptr[i];
    myPort.release ();
 }
}
```

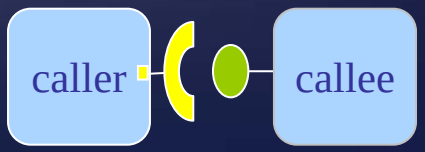
## User View

```
interface ExtendedServices : Services {
  void createAccessPort
                (in string  portName,
                 in string  typeDataName,
                 in TypeMap properties);
  void createSharesPort (…);
}
interface AccessPort : Port {
  opaque get_pointer();
  long get_size();
  void acquire();
  void acquireR();
  void release();
}
interface SharesPort : AccessPort {
  void associate (in opaque ptr,  in long size);
  void disassociate();
}
```

# Data sharing on operation invocation

- **Data sharing in SIDL**
  - Add “&” notation

```
interface compute {
   void foo (in matrix & m1, out matrix & m2);
}
```




- **Orthogonal to parameter modes**
  - In/out/inout determines who is the data “publisher”
  - Example

| | data publisher | data access (R/W) |
|---|---|---|
| in& | caller | all/always |
| out& | callee | callee: always<br>caller: after return |

- **Mapping of arguments to data ports**
  - Example

| | caller | callee |
|---|---|---|
| in& | shares port | accesses port |
| out& | accesses port | shares port |

# Example of use

```
interface compute {
   void foo (in matrix & m1, out matrix & m2);
}
```

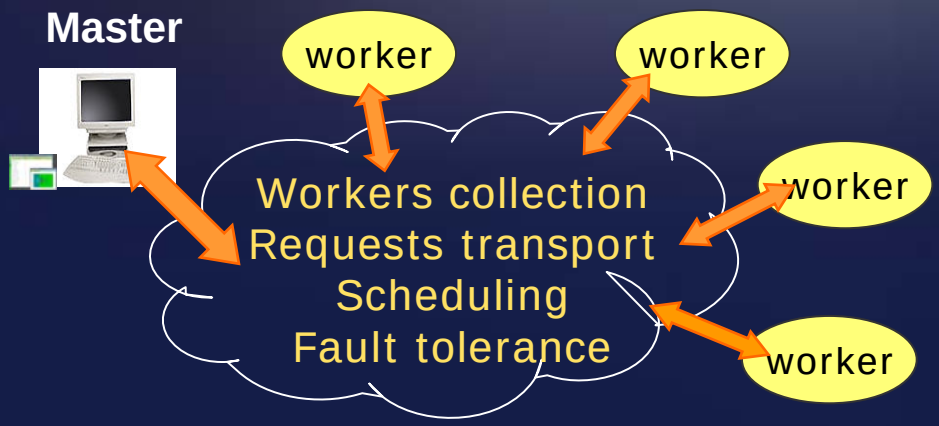

```
Matrix* ptr = allocate_matrix(…)
SharesPort* dp1 =
    createSharesPort(“p1”, “matrix”)
dp1->associate(ptr, size);
AccessPort* dp2 =
    createAccessPort(“p2”, “matrix”);
to_server-> foo(dp1, dp2);
```

```
void foo (AccessPort& dpm1,
          SharesPort& dpm2) {
  ptr = dpm1.get_pointer();
  size = dpm1.get_size();
  res = f77_foo(ptr, size);
  dpm2->associate(res, size);
}
```

# Outline of the talk

- **Introduction**
  - Context
  - Overview of the Common Component Architecture
- **Data sharing paradigm**
  - Objectives
  - Our proposition : data port model
  - Data sharing on operation invocations
- **Support of the master-worker paradigm**
  - Limits with existing component models
  - A proposed model
- **Data sharing in a master-worker paradigm**
- **Conclusions and perspectives**

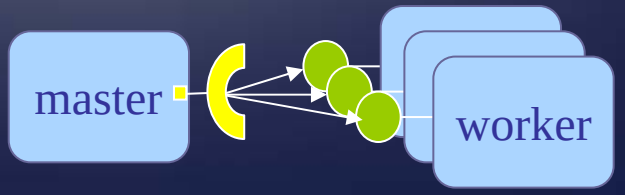
# Problem overview

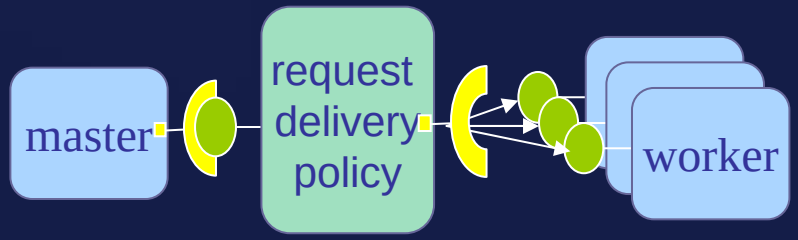



- **Simultaneous independent computations (~ForAll loop)**
- **Dedicated API/environments**
  - BOINC, XTremWEB,
    DIET, NetSolve, Nimrod/G, ...

# Limits with component models

- **Different infrastructures**
  - Multi-core processors, SMP, clusters, grids, etc.
- **Resources dependant properties**
  - Number of workers
  - Request transport and scheduling policy

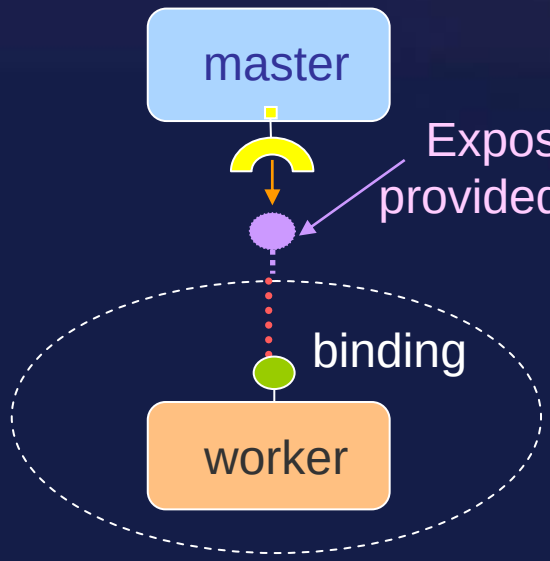



- **At the burden of the programmer**
  - Complex
  - No transparence
- **Objectives**
  - Transparency
  - Re-use existing MW environments

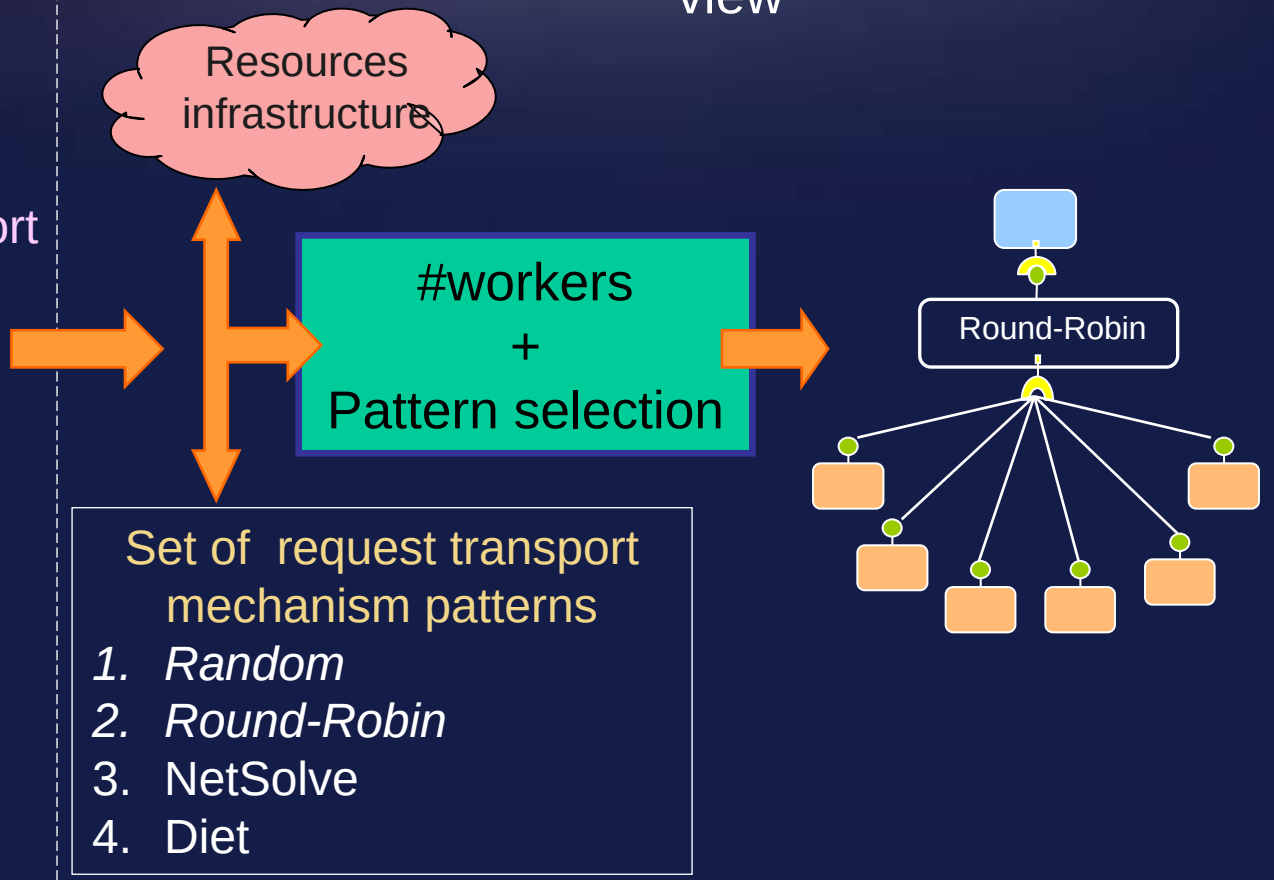
# Overview of the proposal

Programmer/designer view




Collection

Framework implementation view

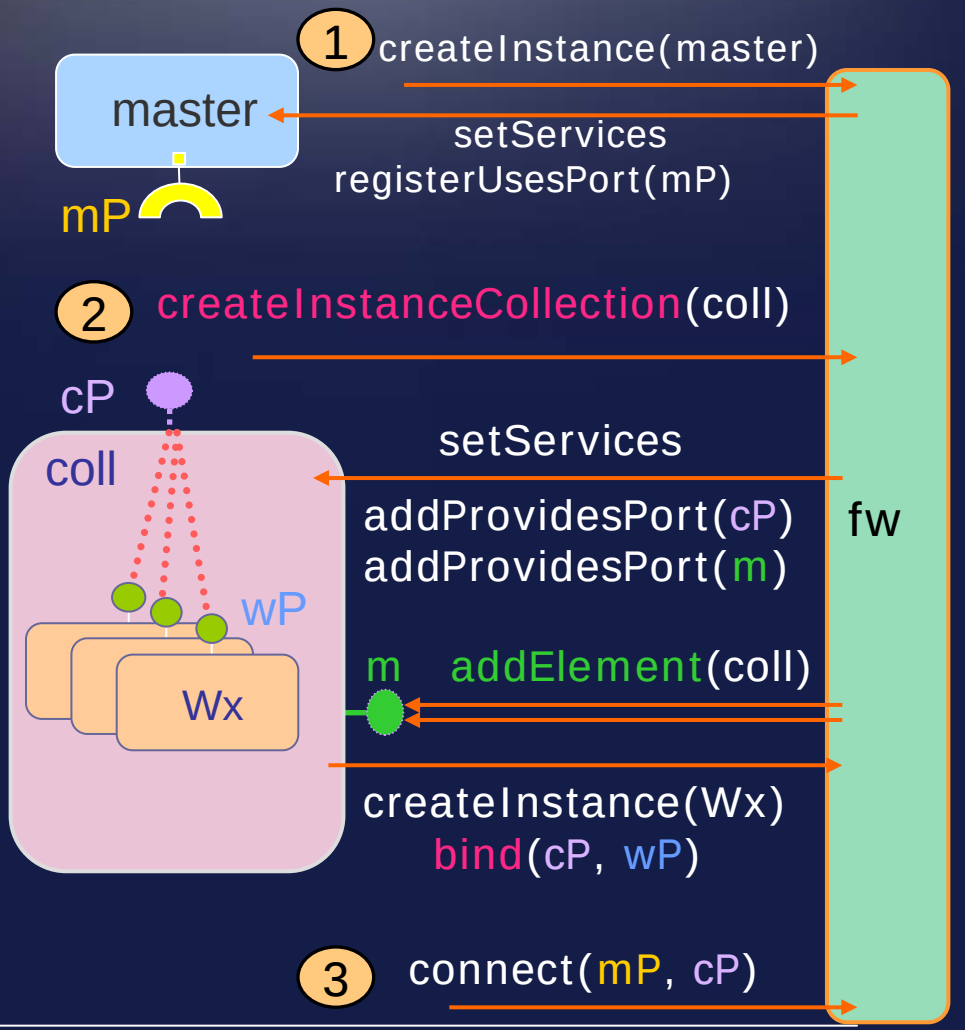



Set of request transport mechanism patterns

1. *Random*
2. *Round-Robin*
3. NetSolve
4. Diet

# Master-worker paradigm in CCA

```
interface CollBuilderService
              : BuilderService {
  CollectionID createInstanceCollection
              (in string instanceCollName,
               in string className,
               in TypeMap properties);
  BindingID bind
              (in CollectionID collID,
               in string collPortName,
               in ComponentID elem,
               in string elemPortName);
}
interface CollectionManagement: Port {
  bool addElement(in CollectionID collID);
  bool subElement(in CollectionID collID);
}
interface PatternInstantiation: Port {...}
```

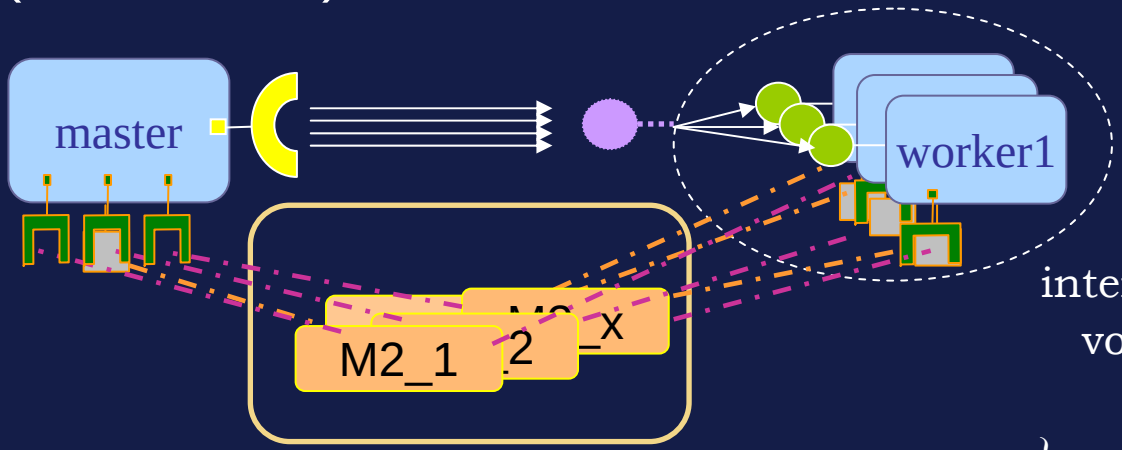


# Outline of the talk

- **Introduction**
  - Context
  - Overview of the Common Component Architecture
- **Data sharing paradigm**
  - Objectives
  - Our proposition : data port model
  - Data sharing on operation invocations
- **Support of the master-worker paradigm**
  - Limits with existing component models
  - A proposed model
- **Data sharing in a master-worker paradigm**
- **Conclusions and perspectives**

# Data sharing on operation invocation

- **Data sharing on operation invocations**
  - Operation declarations level
- **Master-worker paradigm**
  - Composition level
- **Data-sharing & master-worker paradigm**
  - Dynamic data ports creation at each operation invocation (worker side)




```
interface compute {
   void foo (in matrix & m1,
             out matrix & m2);
}
```

# Conclusions and perspectives

- **Data sharing and transparent data access model in CCA**
  - Data ports model: «shares» and «accesses»
- **Improving the support of master-worker paradigm in CCA**
  - Collections, request delivery policy patterns
  - Resources infrastructure independent programming
  - Adaptation of workers and requests delivery at the burden of the framework
- **Support of data sharing on operation invocation**
  - Data sharing in MW paradigm
- **Perspectives**
  - Implementation within a CCA framework
    - Data sharing and master-worker paradigm
  - Benchmarks
  - Shared components in a distributed environment

# Questions ?