High Performance Components with Charm++ and OpenAtom

(Work in Progress)

Christian Perez

Graal/Avalon INRIA EPI

LIP, ENS Lyon, France

Joint Laboratory for Petascale Computing

University of Illinois at Urbana-Champaign

23 November 2010

Context of this work

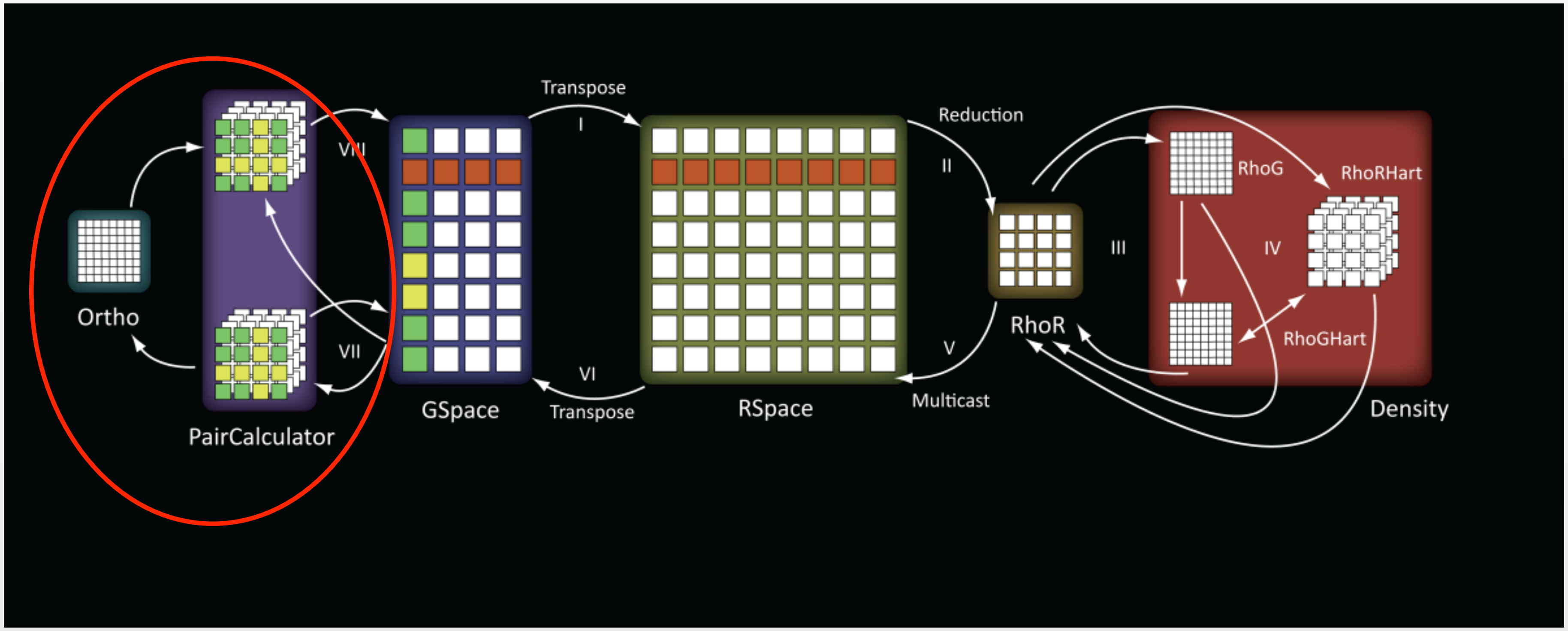
- Initial discussion with **Laxmikant Kale**
  - 2nd workshop, Urbana, 2-4 December 2009
- Actual start
  - Visit of **Julien Bigot** & Christian Perez at UIUC, 19-23 July 2010
  - Fruitful discussion with
    - **Phil Miller** (Charm++)
    - **Eric Bohm** and **Ramprasad Venkataraman** (OpenAtom)

Outline of the talk

- Motivation
  - OpenAtom
- Overview of HLCM core concepts
  - HLCM/Charm++
- Some examples with HLCM
  - Shared Memory
  - MxN
  - Advanced Chooser
- Current status & ongoing work

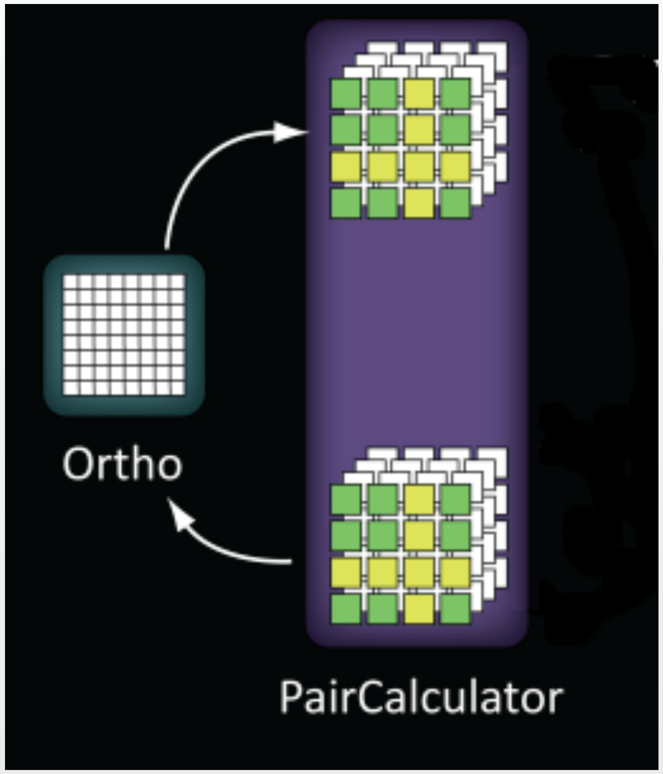
Overview of OpenAtom (UIUC)

- Ab-initio quantum chemistry code based on Charm++



PairCalulator & Orthonormalization Modules

- PairCalculators
  - 4-dimensional (4D) array
  - Used in the force regularization and orthonormalization phases
- Ortho
  - 2D array
- Interactions between PairCalculators & Ortho
  - Specialized Reduction & Multicast based operations



Issues with OpenAtom (PC/Ortho)

- No well defined separation of codes
  - How to replace the Ortho code with an improved version?

- Mixing of concerns
  - PairCalculator & Ortho codes mixed with optimized communication code

- No abstraction for adapting the application (code and performance portability)
  - How to select an Ortho implementation in function of hardware and input data?
  - How to select an optimized communication implementation between PC & Ortho?

Objectives

- Enable code-reuse
  - E.g. the Ortho module of OpenAtom
  - Let expert develop a piece of code
- Enable *adaptation* when re-using code
  - E.g. should Ortho be based on double? What about quad?
  - Let re-use code with parameterization options
- Enable any kind of composition operators
  - E.g the 4D-2D interactions between PairCalculator & Ortho
  - Do not impose any communication models
- Enable efficient implementation of composition operators
  - E.g. by having a 4D-2D op. instead of reduction+multicast op.

How to Achieve Those Objectives?

- Enable code-reuse
  - Software Component
    - Primitive component for re-using implementation code
    - Composite component for re-using assemblies of components
- Enable *adaptation* when re-using code
  - Genericity
- Enable any kind of composition operators
  - Connectors
- Enable efficient implementation of composition operators
  - Open connection

Overview of Core Concepts of High Level Component Model (HLCM)

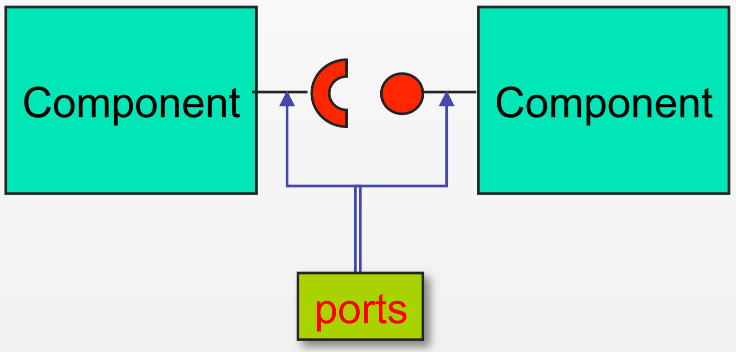
Component, Connector, Hierarchy,
Genericity, & Template Meta-Programming

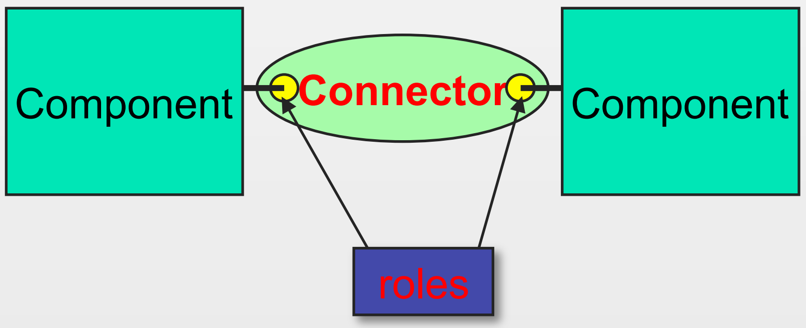
HLCM: High Level Component Model

- Defined in the PhD of Julien Bigot
- Major concepts
  - Component model
    - Primitive (abstract) and composite
  - Connector based
    - Primitive and composite
  - Generic model
    - Support meta-programming (template *à la* C++)
  - *Currently static*

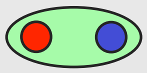
Connectors

- Without connectors
  - Direct connection between ports through model provided interactions
- With connectors
  - Originally defined in ADLs
  - Connectors reify connections
    - A name
    - A set of roles
  - Any number of roles
  - Can be 1st class entities
    - Provided by the underlying model
    - User implemented





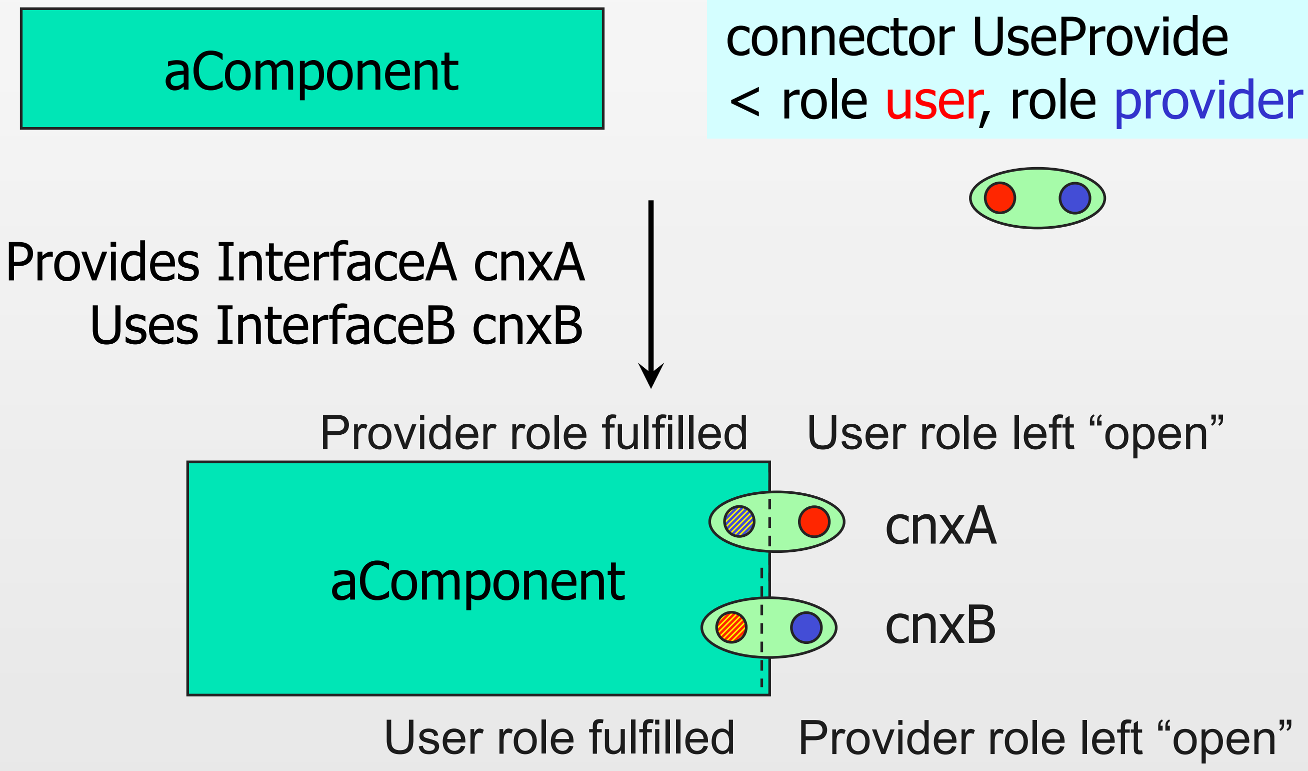
```
connector UseProvide
< role user, role provider >;
```



HLCM: Component

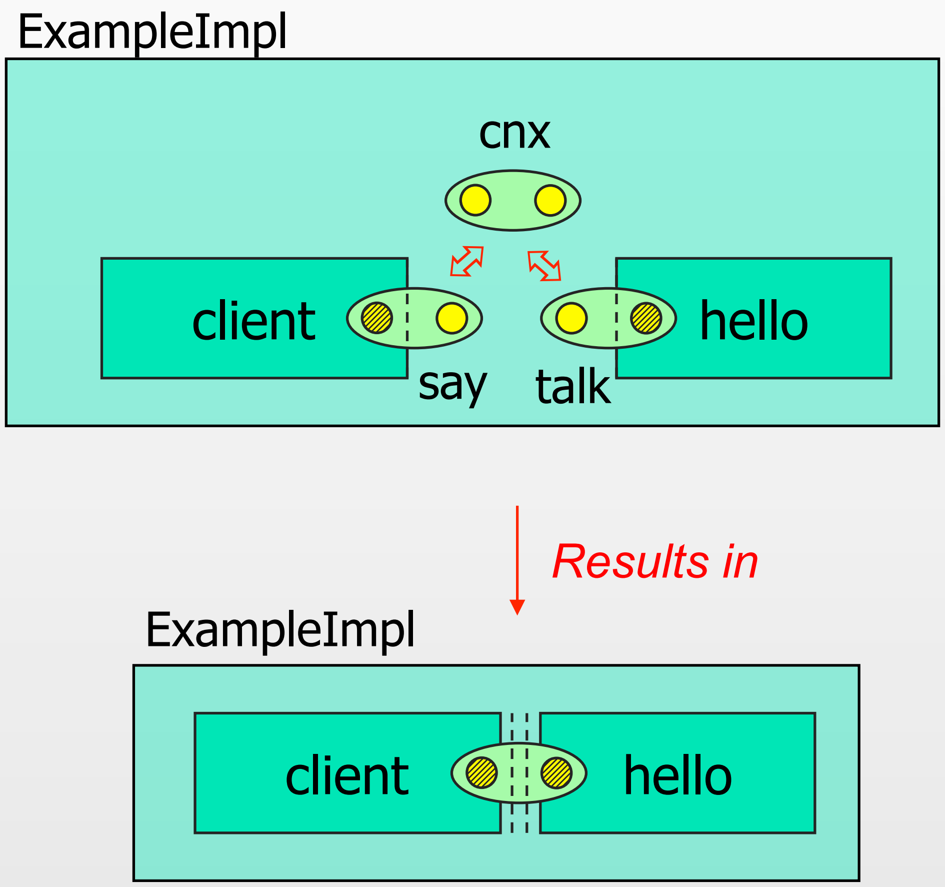
- Black box that may expose some open connections

connector UseProvide
< role user, role provider >;



HLCM: Composite Component

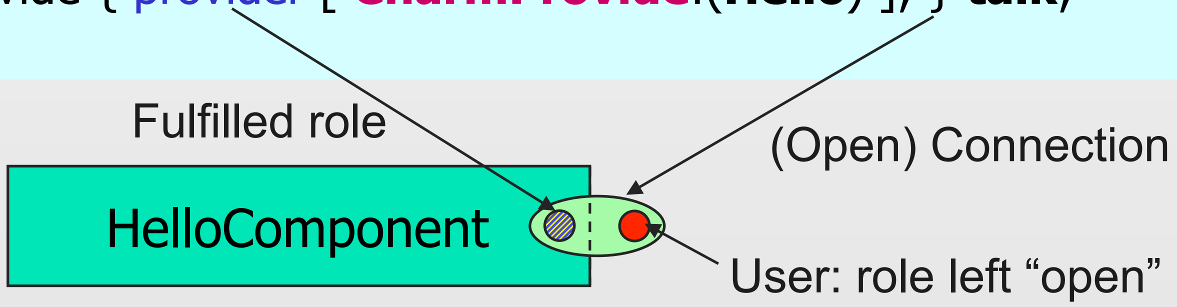
```
component  Example { }

composite  ExampleImpl
implements  Example
{
   HelloComponent hello;
   ClientComponent client;

   connection cnx;
   cnx |= hello.talk;
   cnx |= client.say;
}
```



HLCM: Primitive Components

- Abstract Component Model
  - Primitive components not defined directly by HLCM
  - Primitives defined by a specialization
    - HLCM/CCM, HLCM/Charm++
- HLCM/Charm++
  - Primitive component: Charm++ Chare + some design constraints
  - Primitive connector: UseProvide interactions
    - A chare may provide an interface or make use of a (remote) interface

```
component HelloComponent {
  UseProvide { provider [ CharmProvide!(Hello) ]; } talk;
}
```



HLCM/Charm++ (Nov. 2010)

HLCM

```
component  HelloComponent {
   UseProvide { provider [ CharmProvide!(Hello) ]; } talk;
}
```

Charm++ Primitive Declaration

```
chare HelloC implements HelloComponent {
   exports talk type=Hello as talk.provider;
}
```

Charm++

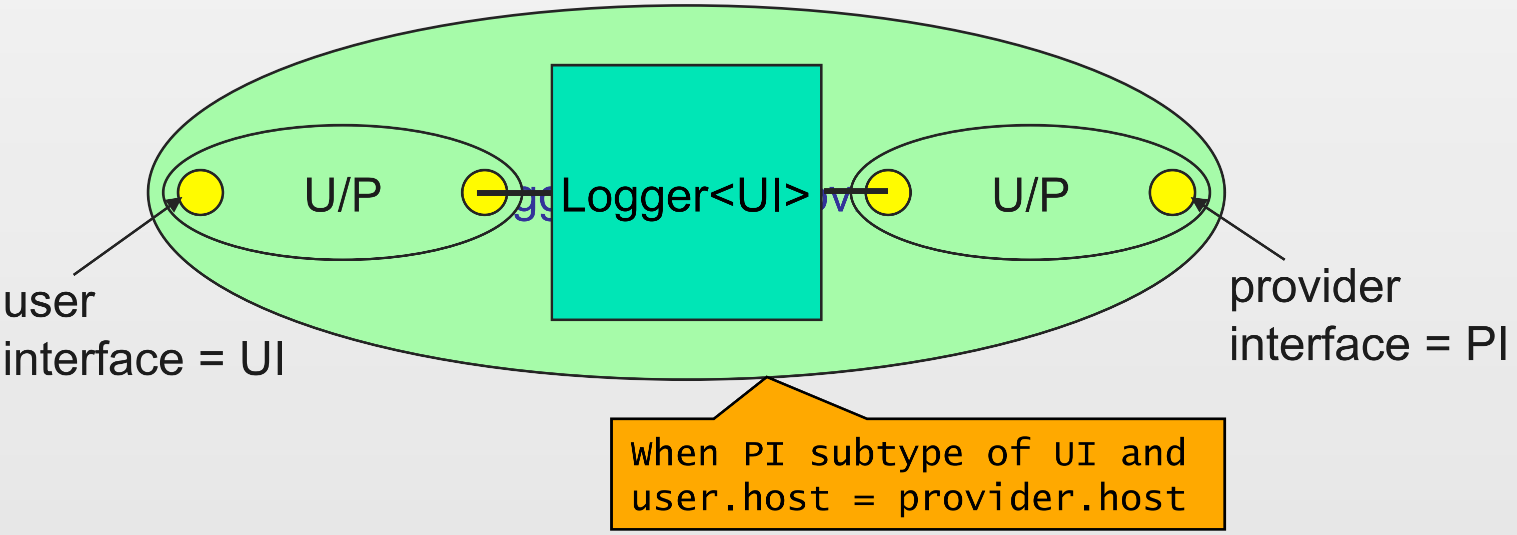
```
chare HelloC : ComponentInterface, Hello {
   entry HelloC();
   // Hello Interface  Implementation (functional code)
   entry void hello() { CkPrintf("Hello!\n"); }
   // Provides Hello talk (could be generated)
   entry [sync] void provider_set_talk(CProxy_ComponentInterface& pssi,
                                        int n, char name[n], long key)
   { pssi._set(thishandle, n, port, key); }
}
```

Engineering issues with HLCM/Charm++

- Need multiple inheritance
  - Implemented by the Charm++ team during summer 2010
  - Validated with components providing 3 interfaces
- Engineering issue with application linkage
  - Charm needs to know all chares to generate stubs
    - Prevent dynamic loading of components
  - Current solution: statically list all used components in Makefile

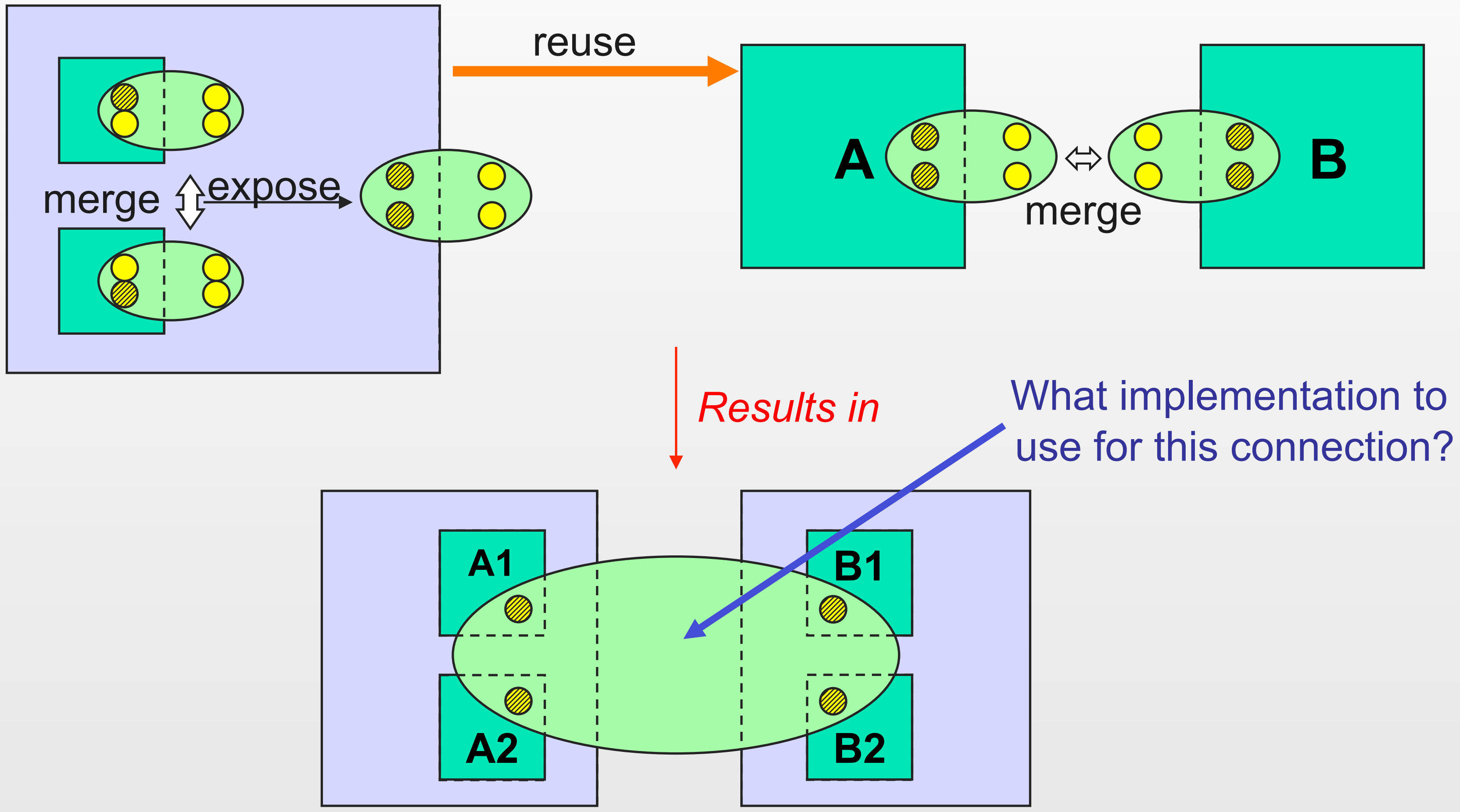
HLCM: User Implemented Connector

```
generator   LoggingUP<UI,PI>
Implements  UseProvide<provider = { CharmProvide!<PI> },
                       user     = { CharmUse!<UI>      }>
when ( UI super PI )
{
   Logger<UI> proxy;
   proxy.clientSide.user += this.user;
   proxy.serverSide.provider += this.provider;
}
```

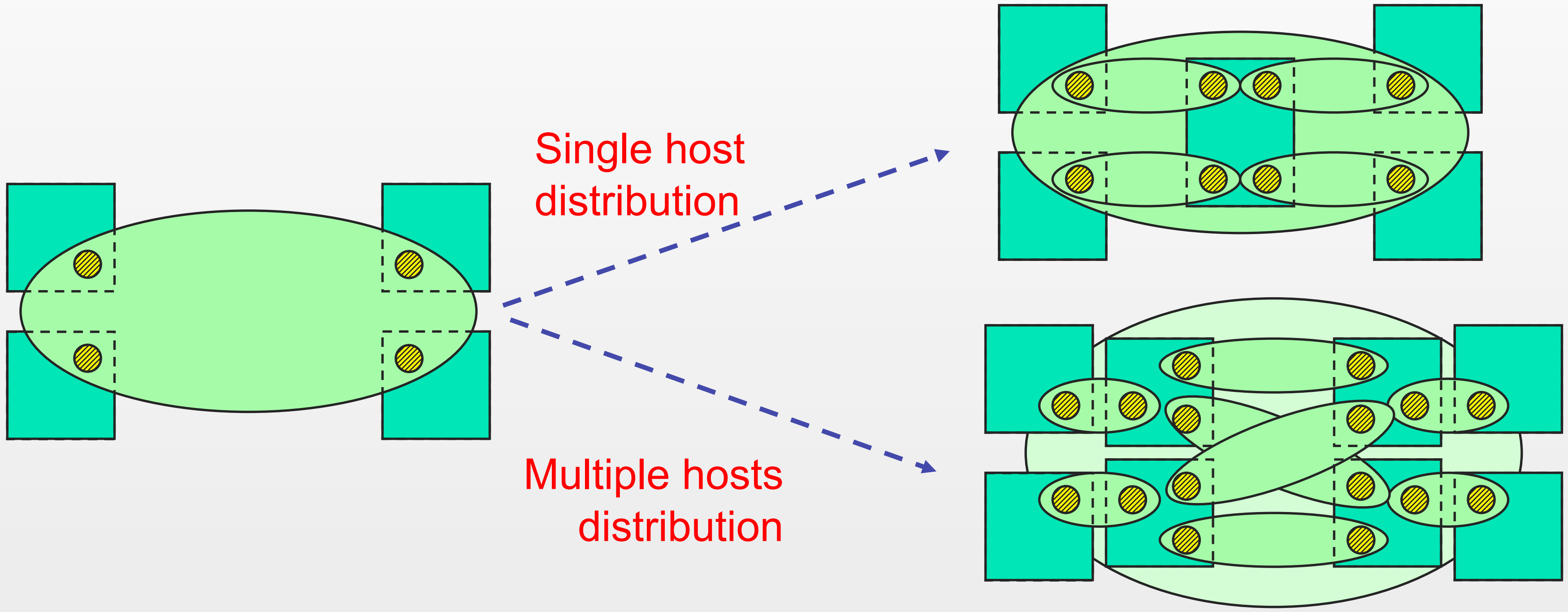


when PI subtype of UI and
user.host = provider.host

HLCM: Benefit of Open Connections



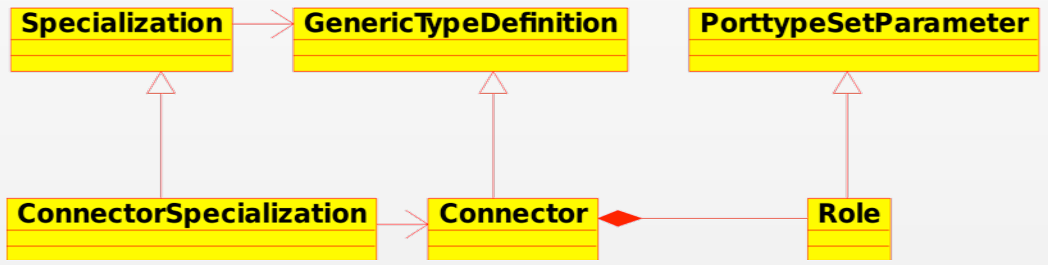
HLCM Connection Implementation: a Planning Choice



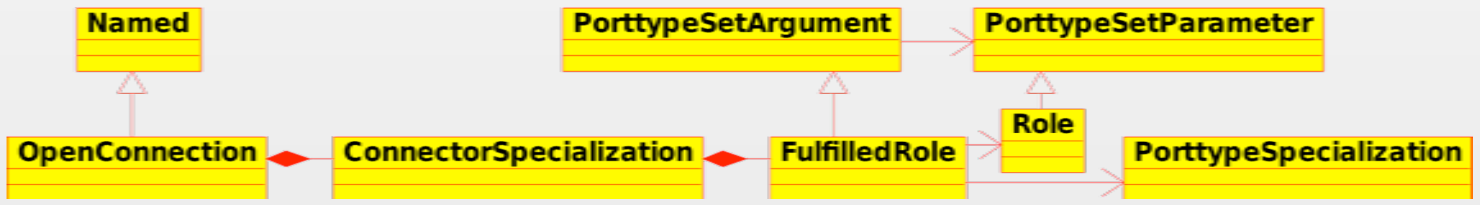
- Component and connection implementation choice made by *choosers*
  - Not defined in HLCM
  - Specialization depend

Model based HLCM Definition

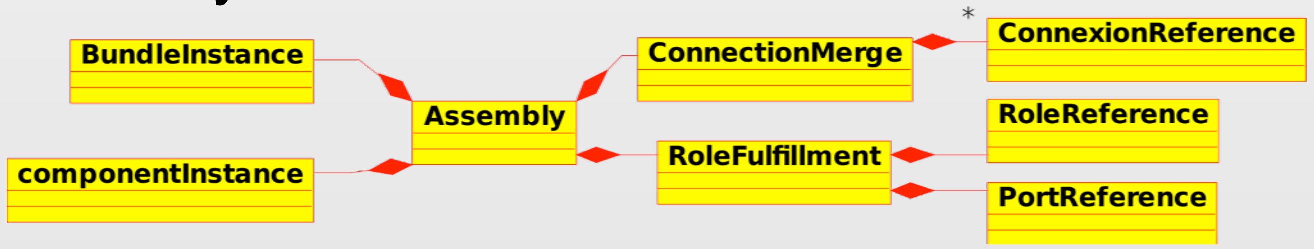
- Connector



- Connection



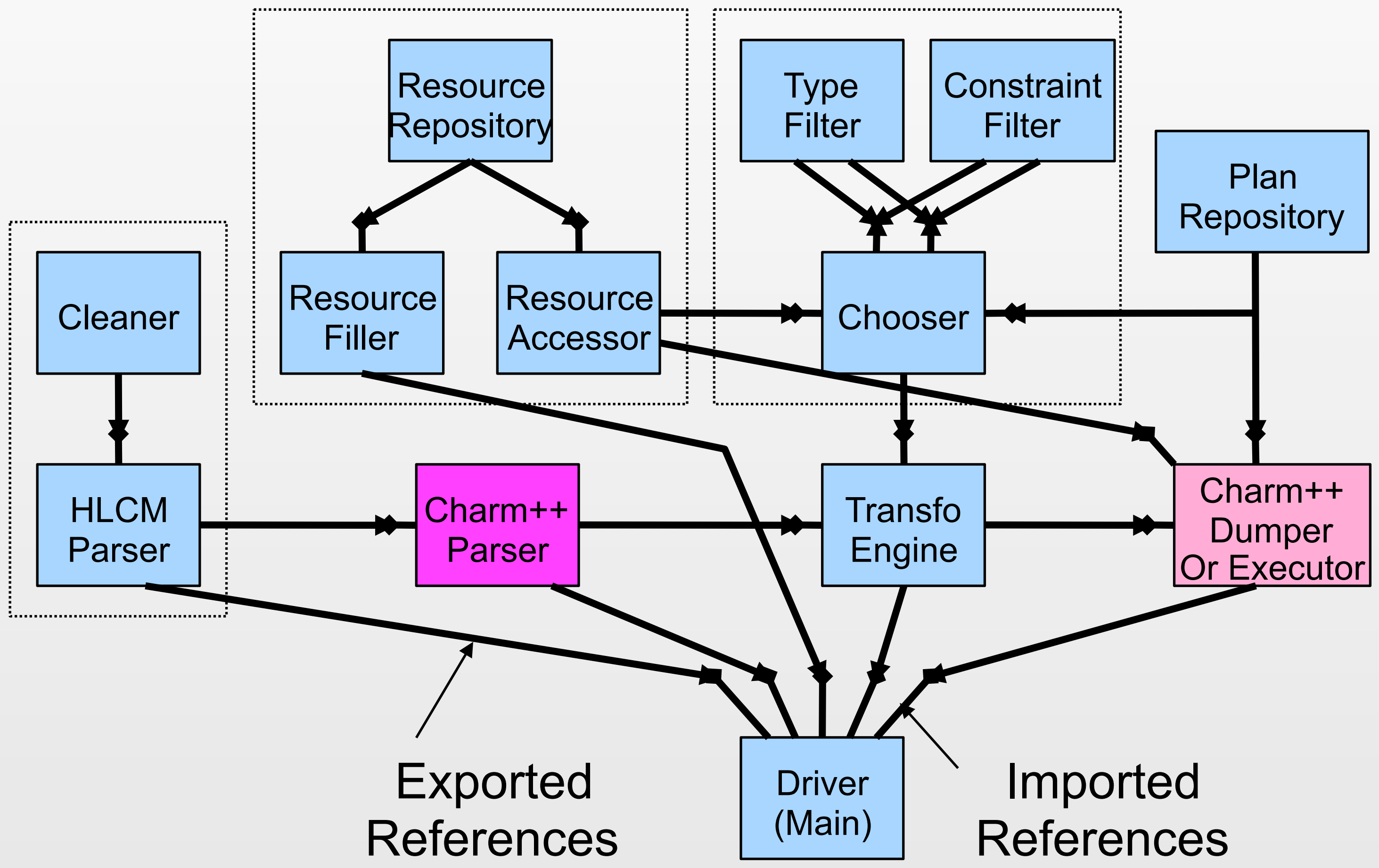
- Assembly



HLCMi: An Implementation of HLCM

- Model-transformation based
  - Eclipse Modeling Tools
  - Mainly Emfatic files
    - Used to generate ecore & Java files
- HLCM core (PIM + transformation)
  - 127 UML classes
  - 470 Emfatic lines
  - 25 000 generated Java lines
  - + 2000 Java lines for transformation engine
    - OMG QVT was not well implemented
- Already implemented connectors
  - Use/Provide, Shared Data, Collective Communications, “MxN” RMI, Irregular Mesh

Architecture of HLCMi/Charm++ in LLCMj



Example of HLCM

Shared Memory

MxN Communications

Hierarchical CEM Application

Example of HLCM

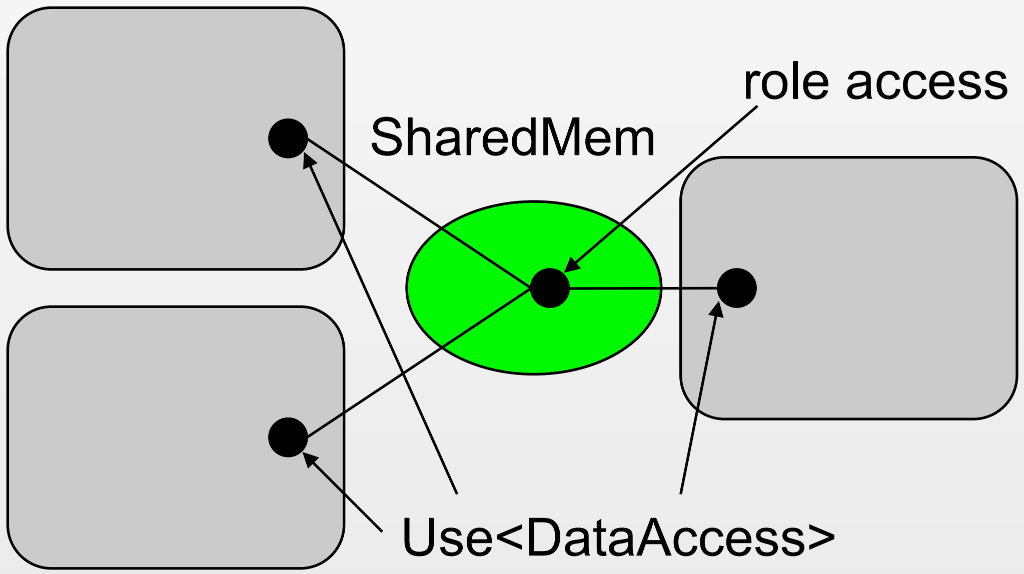
Shared Memory

MxN Communications

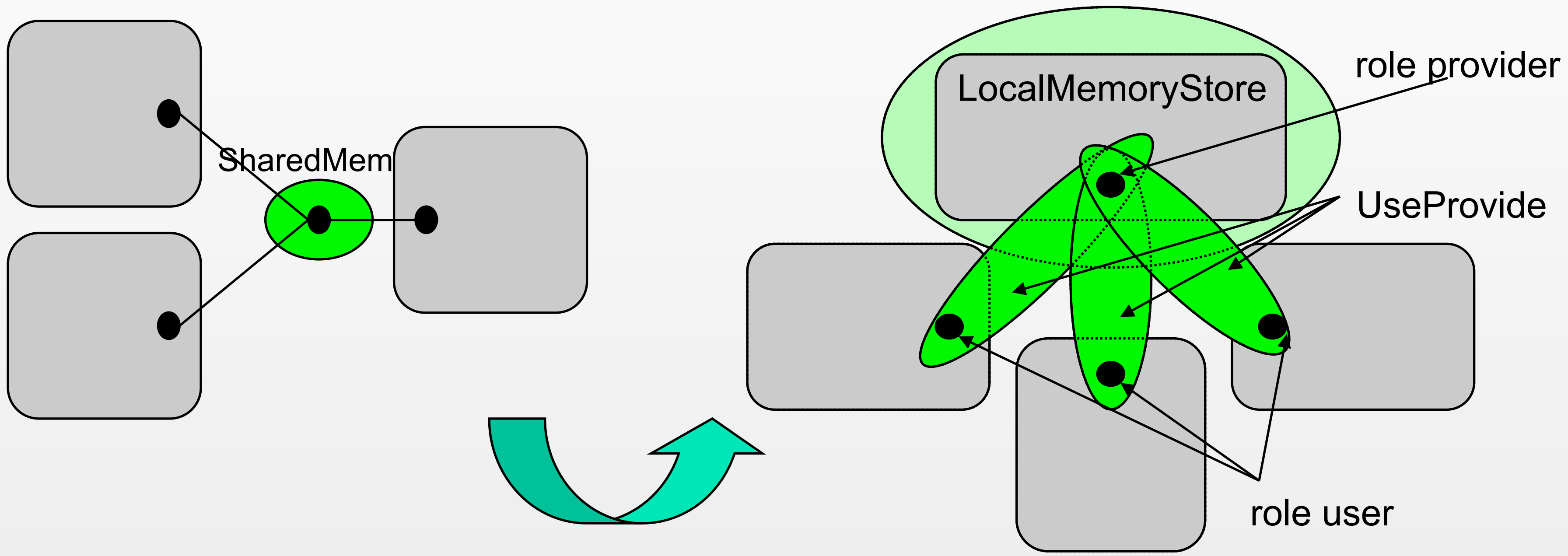
Advanced Chooser

Shared Memory Connector

```
connector SharedMem<role access>;
```



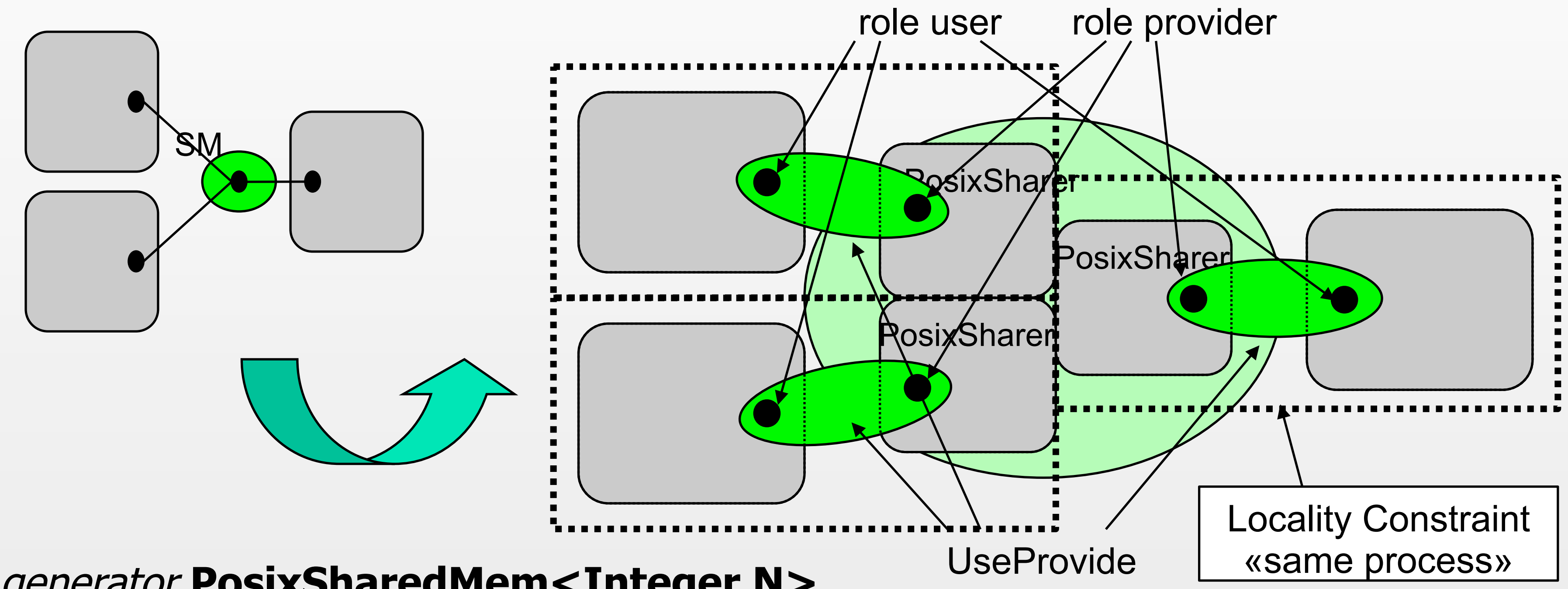
Shared Memory Connector Implementation for Intra-Process Components



generator **LocalSharedMem<Integer N>** *implements*
SharedMem<access=each (i:[1..N]){ LocalReceptacle<DataAccess> } >

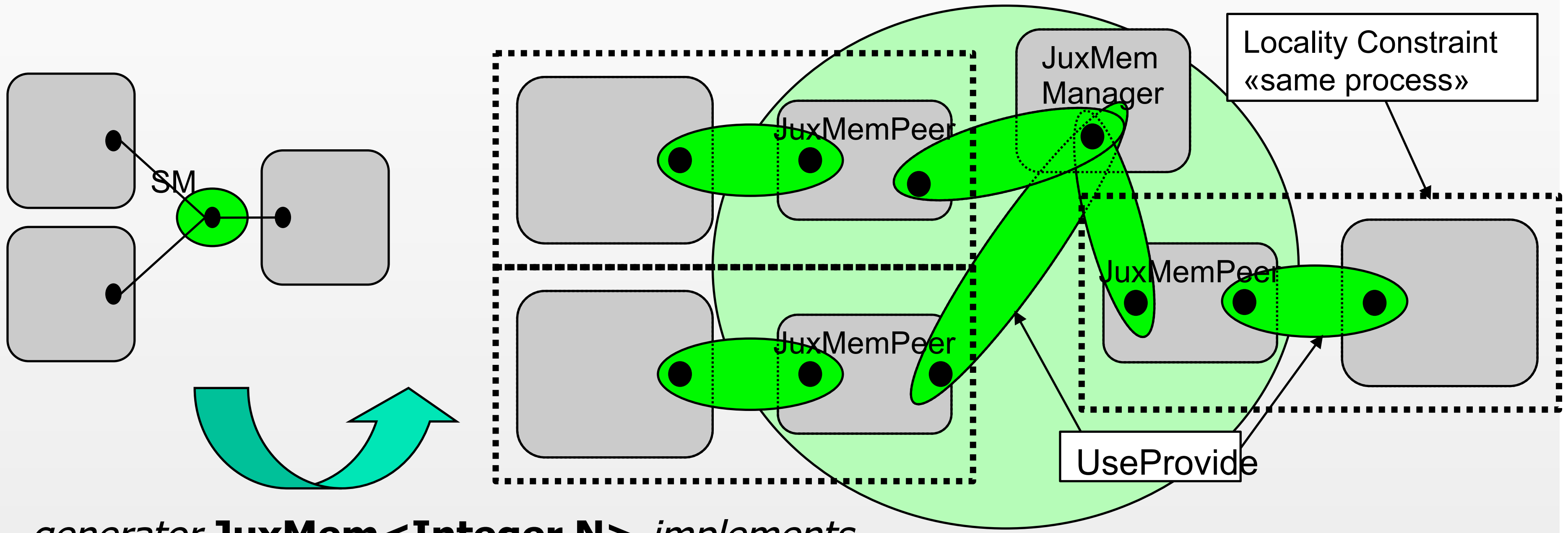
```
{
   LocalMemoryStore<N> store;
   each (i:[1..N]) {
      store.access[i].user+=access[i];
} }
```

Shared Memory Connector Implementation for Inter-Processes, Intra-Node Components



```
generator PosixSharedMem<Integer N>
implements  SharedMem<access=each(i:[1..N]){LocalReceptacle<DataAccess>}>
when samesystem(each (i:[1..N]){ this.access }) {
  each (i:[1..N]) {
    PosixSharer node[i];
    node[i].access.user += this.access[i];
} }
```

Shared Memory Connector Implementation for Inter-Processes, Inter-Node Components



```
generator JuxMem<Integer N> implements
SharedMem<access=each(i:[1..N]){LocalReceptacle<DataAccess>}> {
  JuxMemManager<N> manager;
  each (i:[1..N]) {
    JuxMemPeer peer[i];
     peer[i].access.user += access[i];
     merge ({peer[i].internal, manager.internal[i]});
} }
```

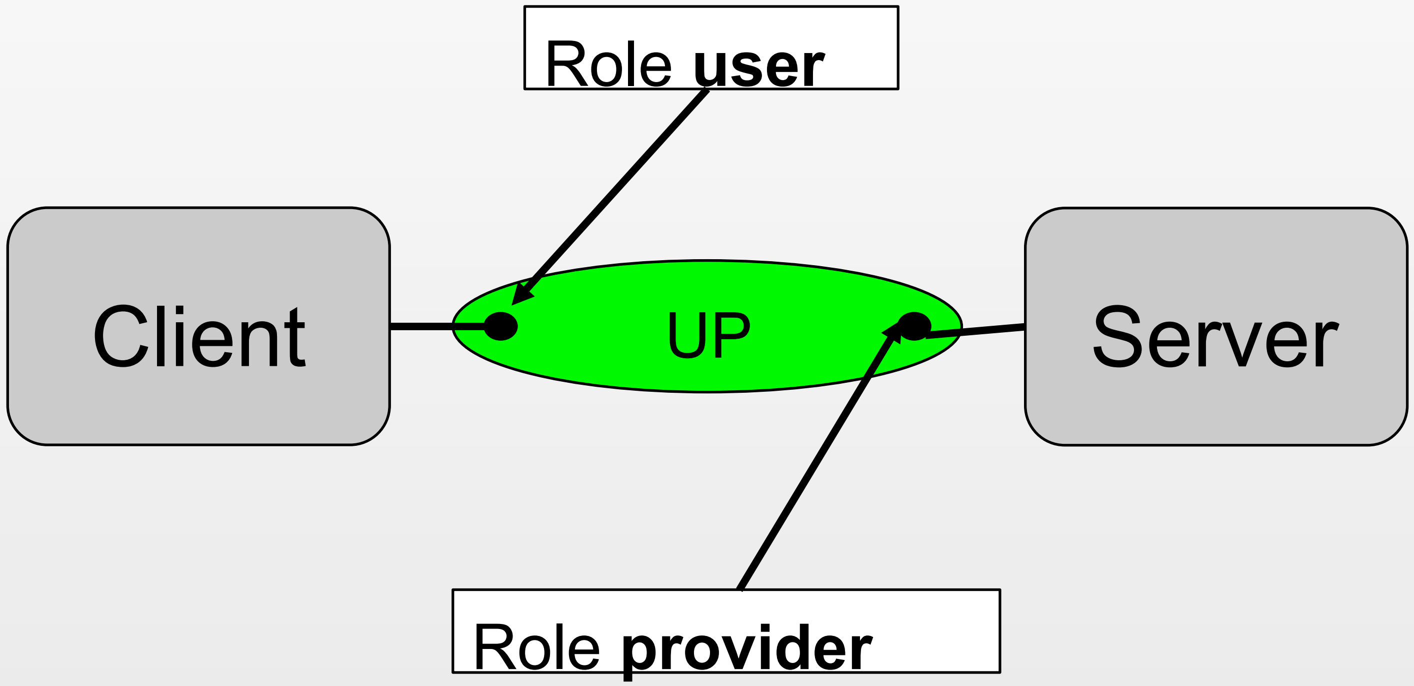
Example of HLCM

Shared Memory

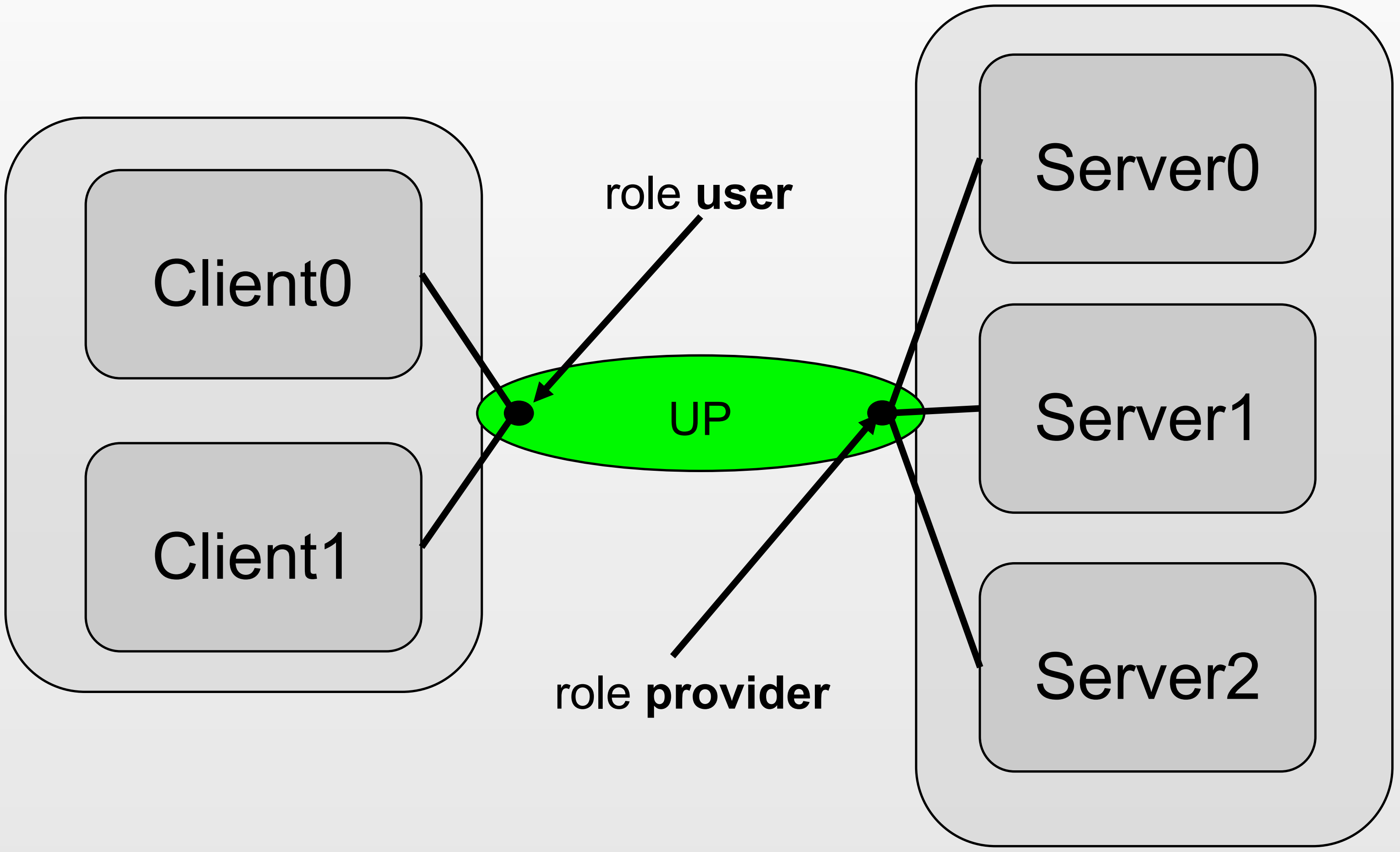
MxN Communications

Advanced Chooser

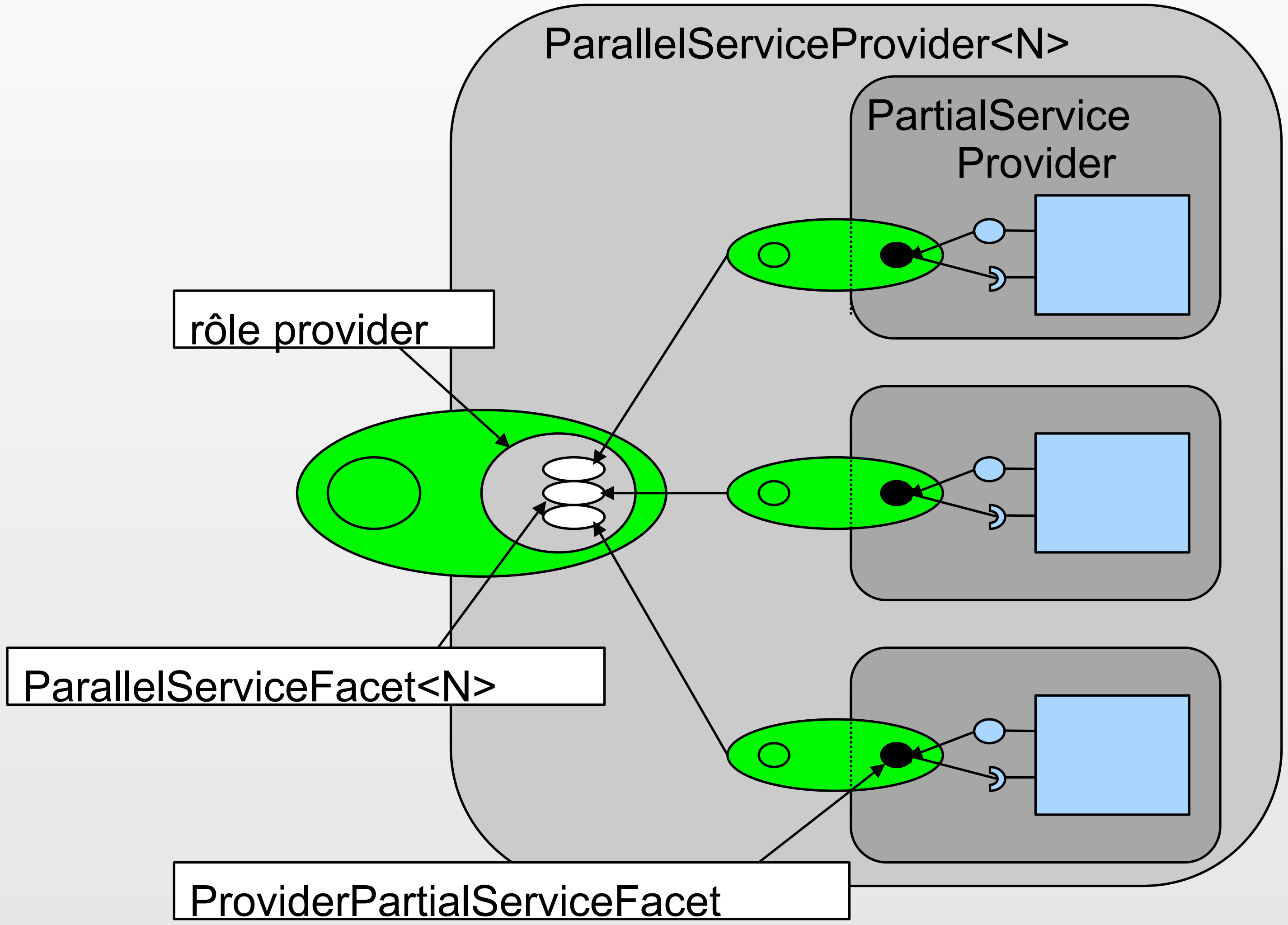
Parallel Components & UseProvide Connector



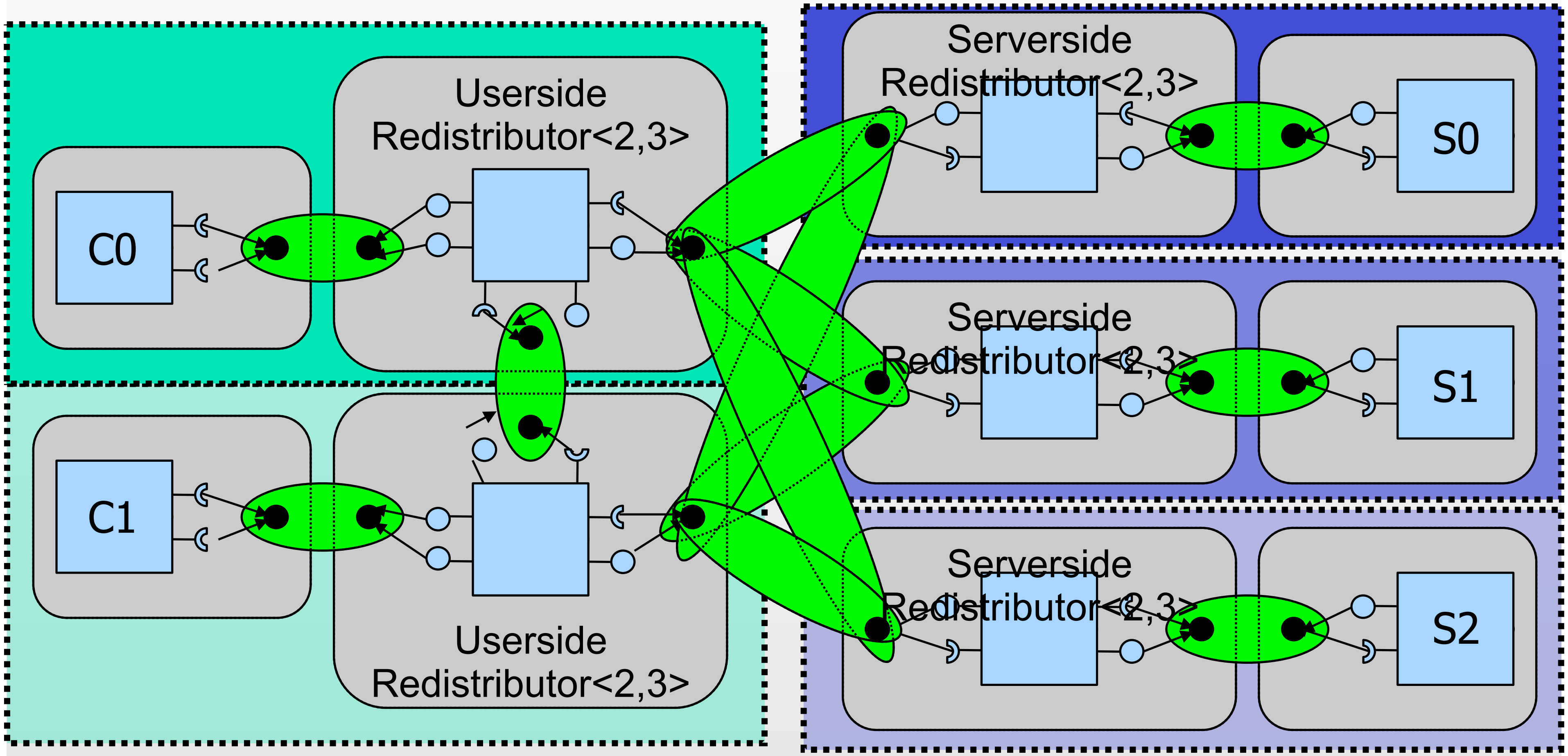
Parallel Components & UseProvide Connector



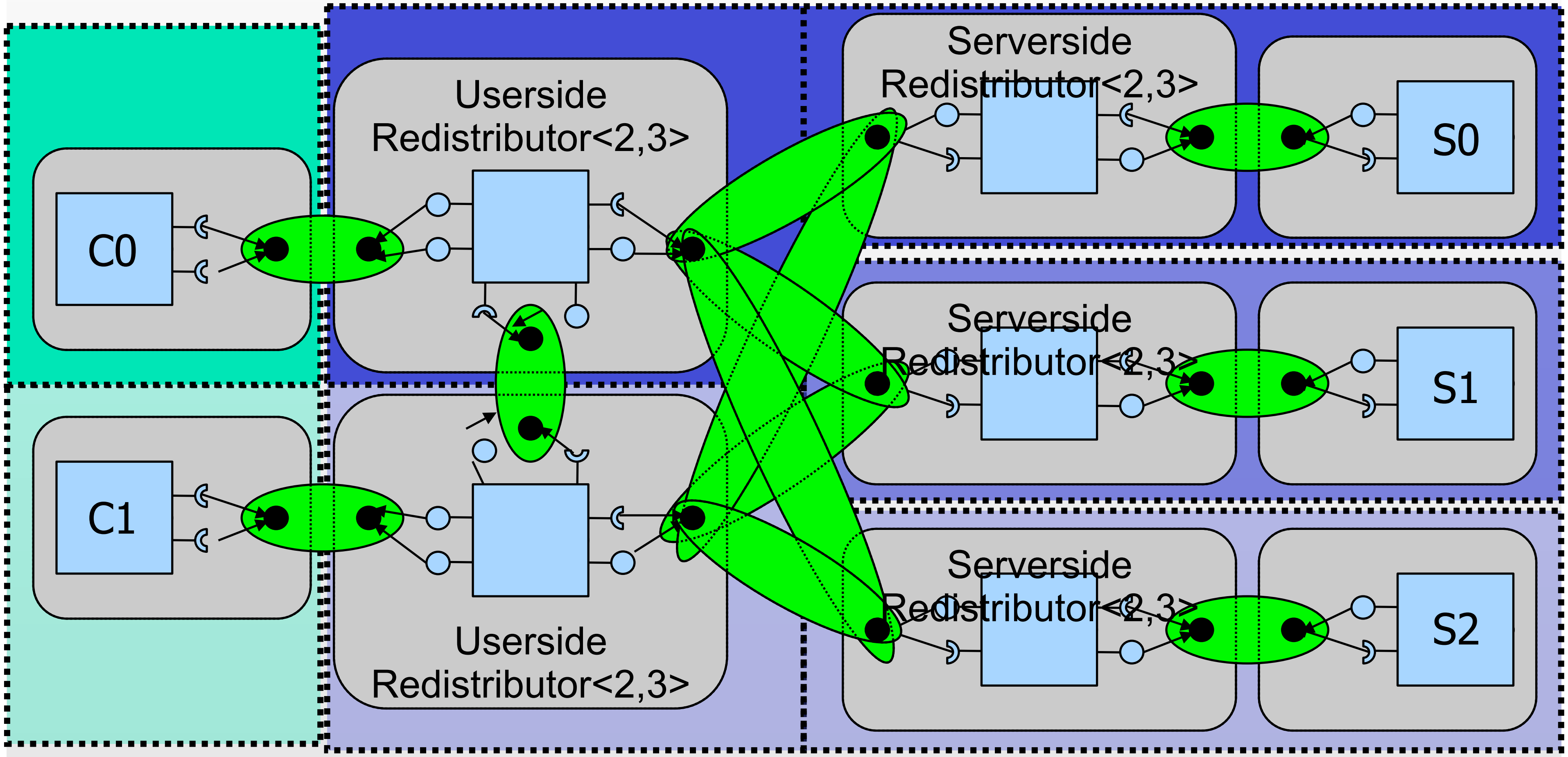
Parallel Components & UseProvide Connector



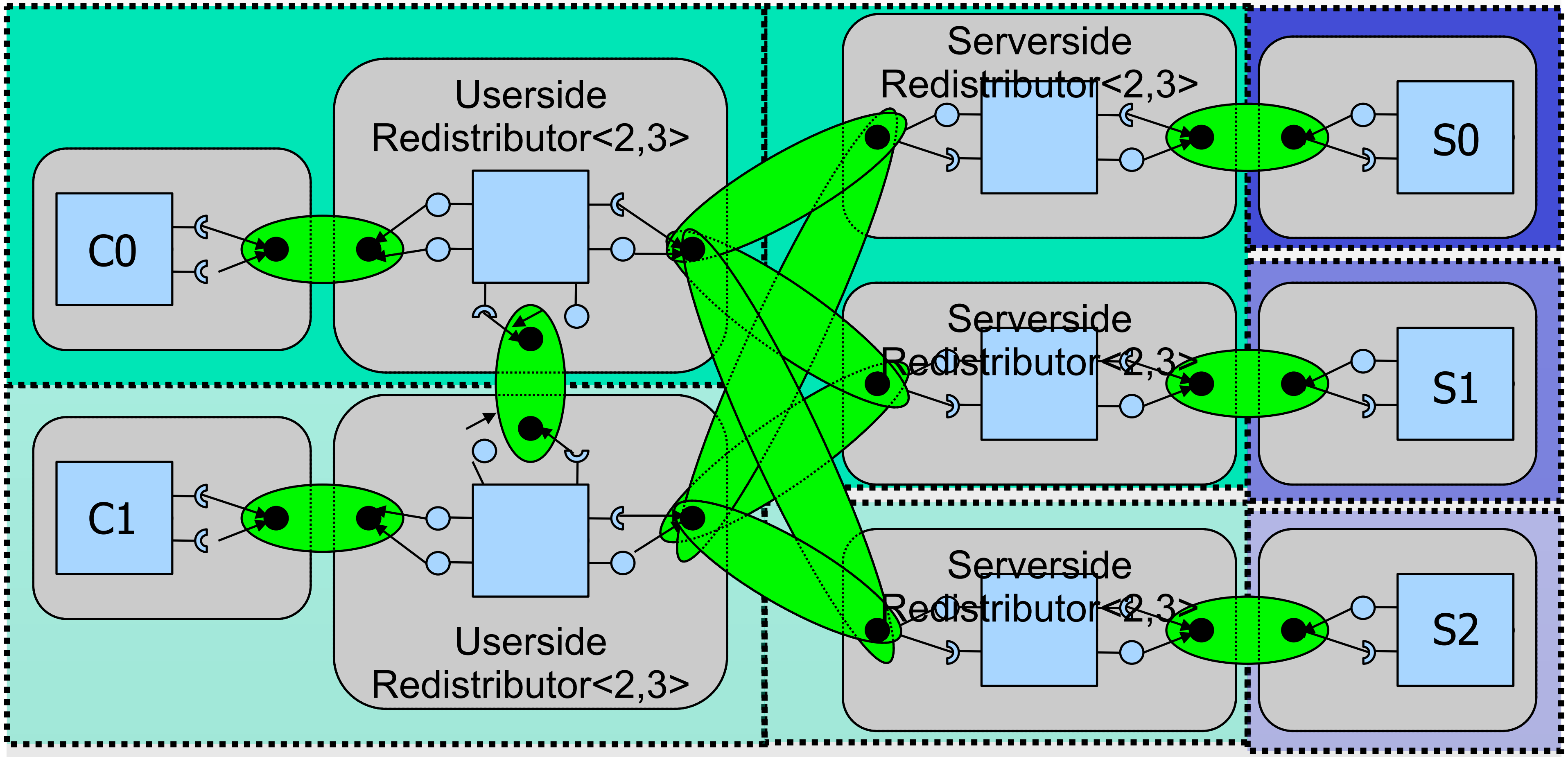
Parallel Components & UseProvide Connector


Userside Redistributor<2,3>
C0
C1
Userside Redistributor<2,3>
Serverside Redistributor<2,3>
S0
Serverside Redistributor<2,3>
S1
Serverside Redistributor<2,3>
S2

Parallel Components & UseProvide Connector



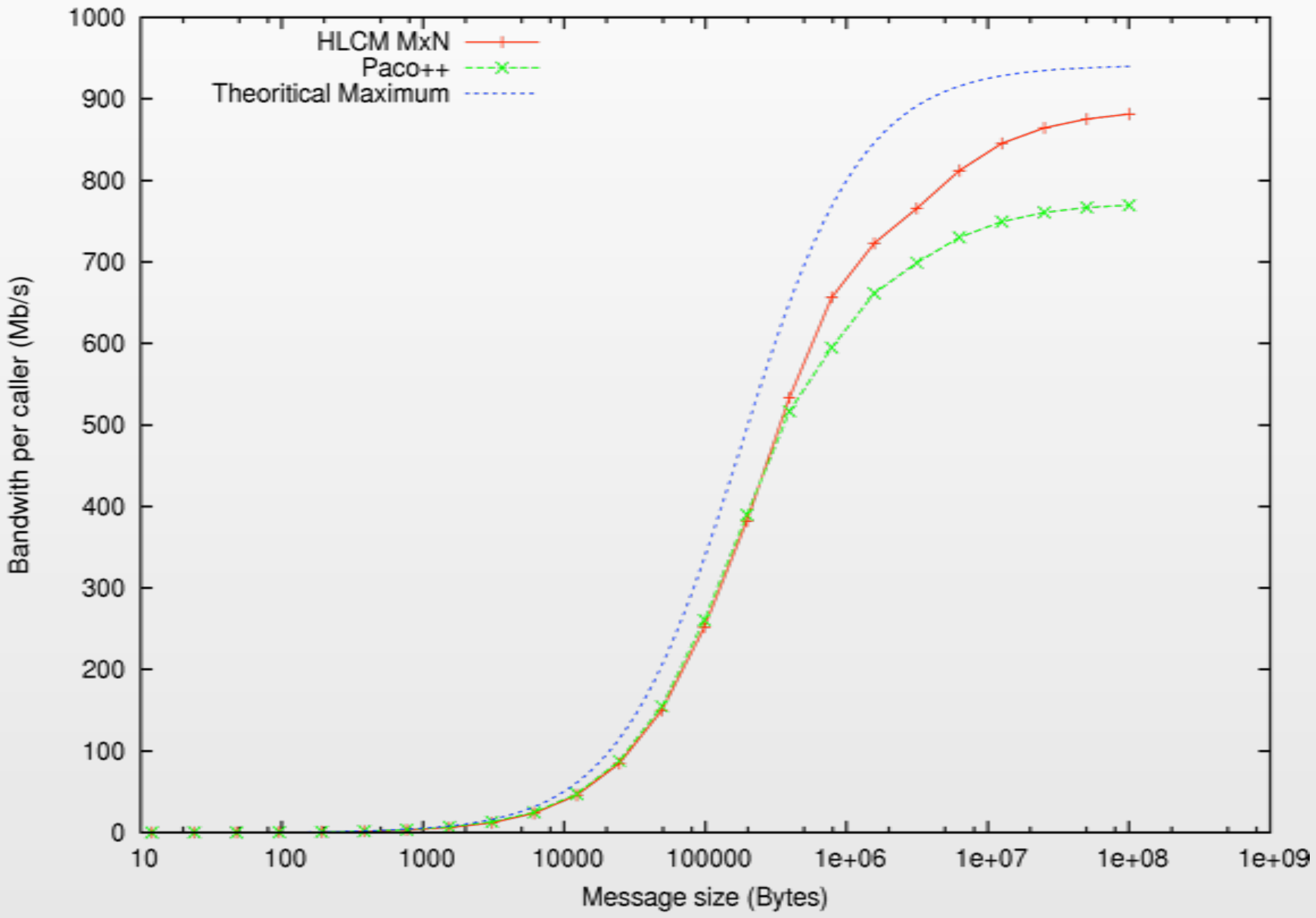
Parallel Components & UseProvide Connector



Parallel Components & UseProvide Connector

- HLCM/CCM
  vs
  PaCO++

- Cluster
- 1 Gbs Ethernet
- HLCM/CCM
- #Client=3
- #Server=4



Example of HLCM

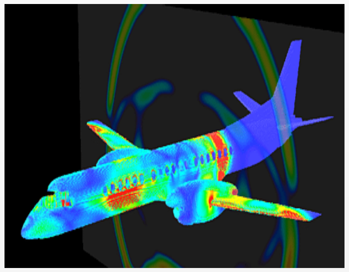
Shared Memory

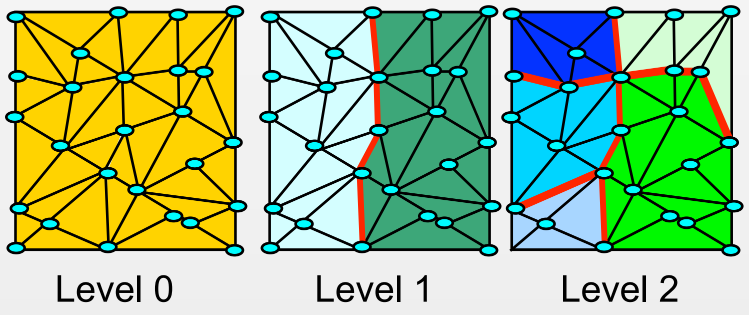
MxN Communications

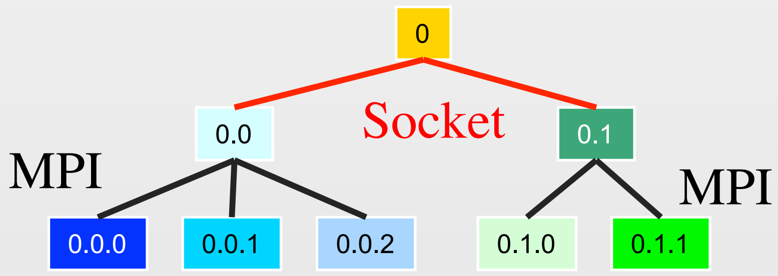
Advanced Chooser

Hierarchical Programming Model

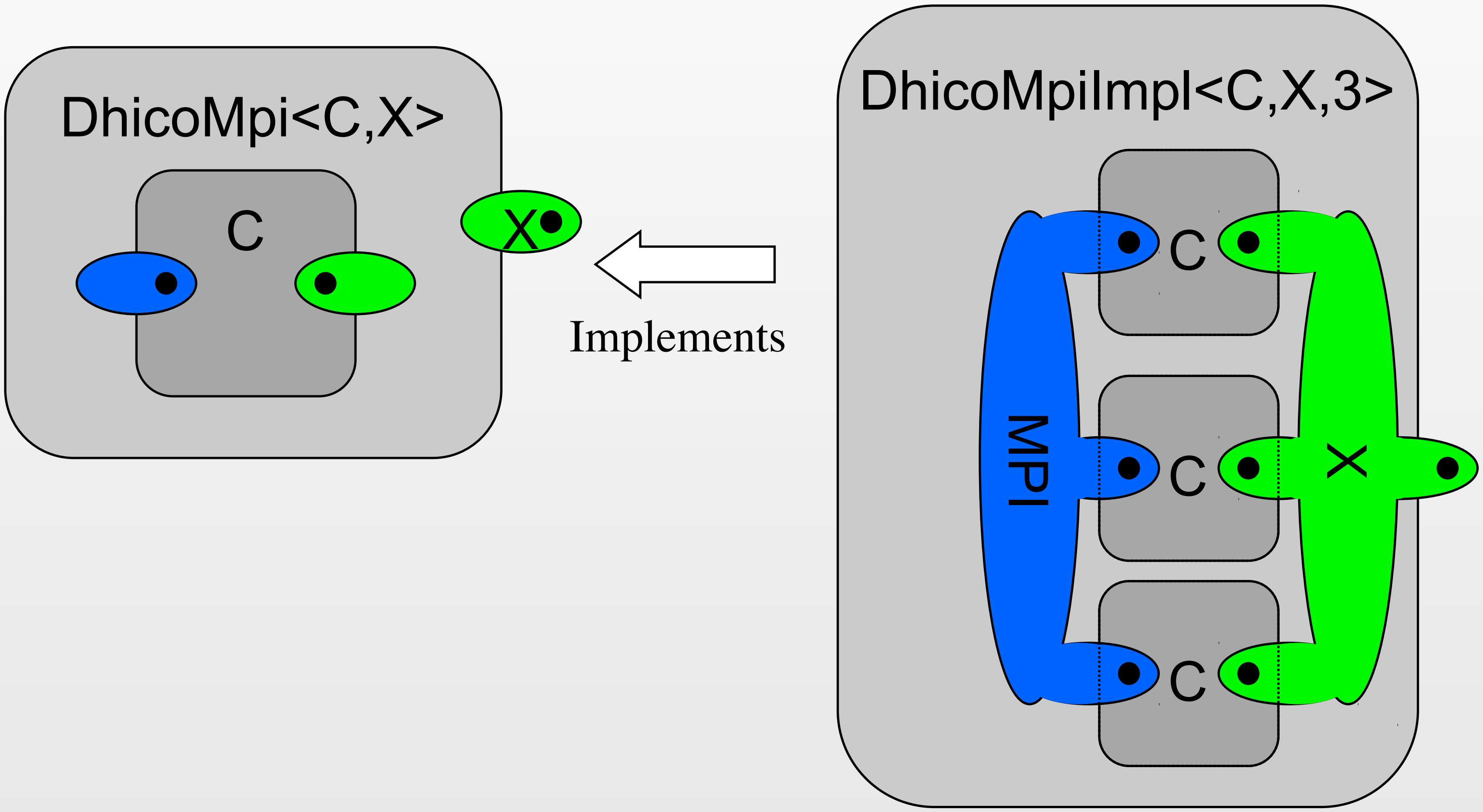
- Application model
  - Moldable, non evolving applications
    - Grid-enabled CEM application
      - French ANR DISCOGRID
      - Set of MPI-based codes
        - How many groups?
        - Size of each group?
- Resource model
  - Hierarchical machines
    - Federation of clusters
- Resource selection
  - Application-specific heuristic available [CKP'09]



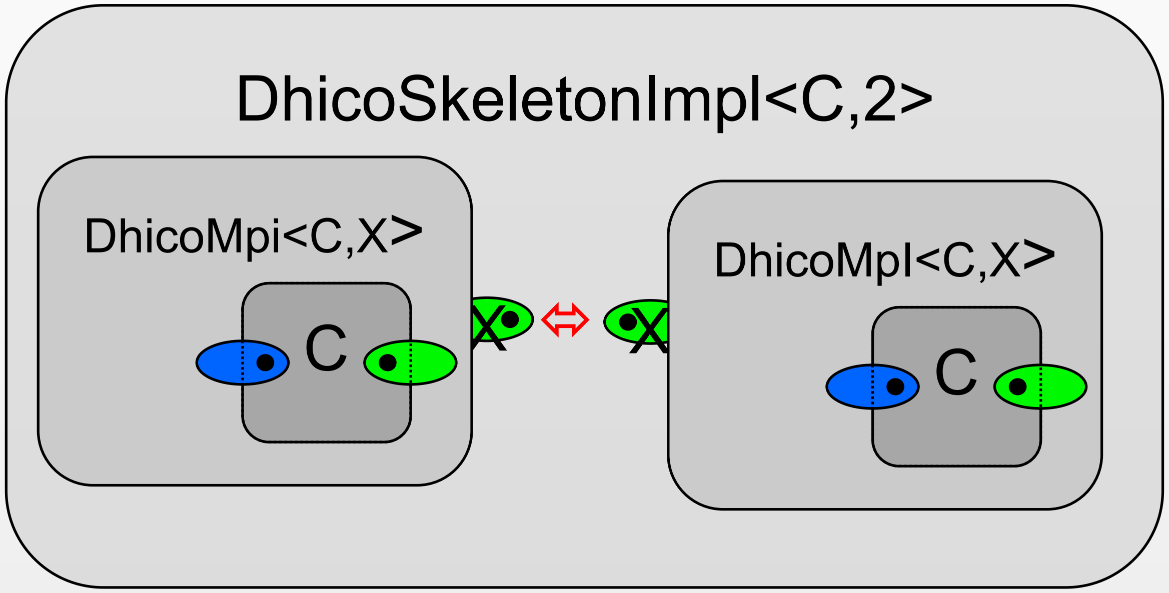




HLCM: Hierarchical Programming Model

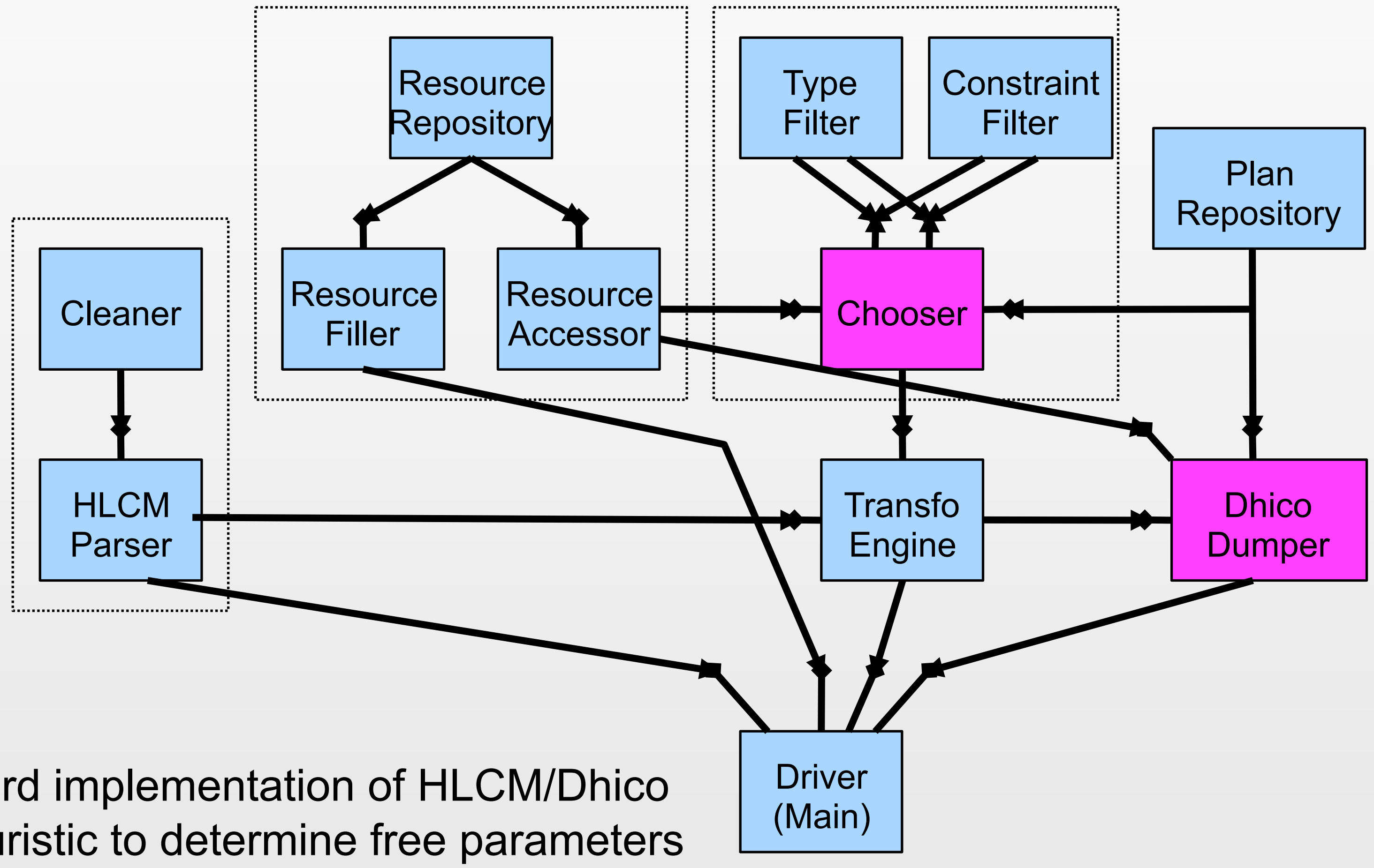


HLCM: Hierarchical Programming Model



```
*composite* **DhicoSkeletonImpl<component C, Integer N>**
*implements* **DhicoSkeleton**<C> {
   each (i:[1..N]){
      **DhicoMPI**<C, **SocketConn**> inst[i];
    }
   merge (each(i:[1..N]){ inst[i].exposed });
}
```

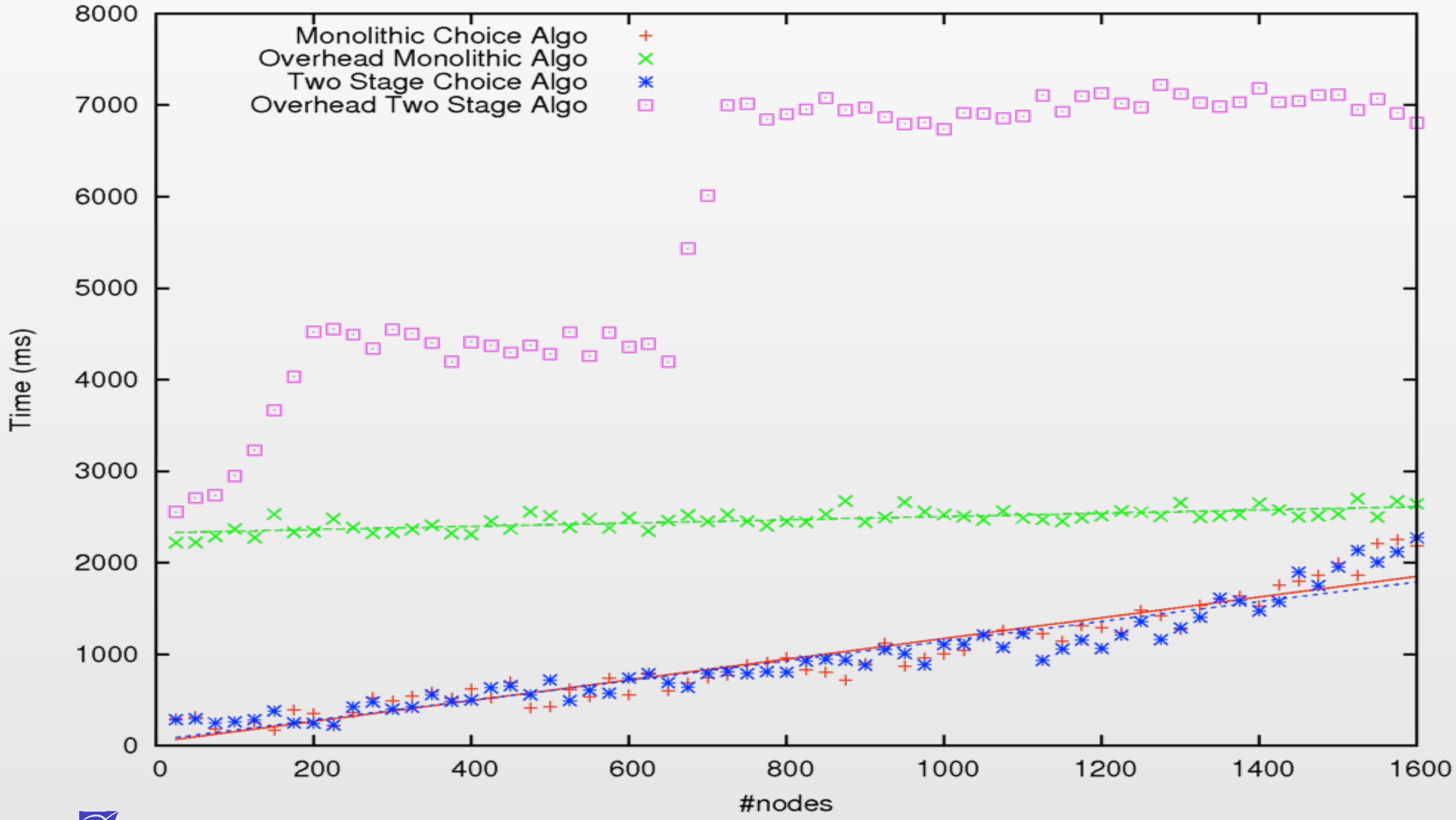
Architecture of HLCMi/Dhico in LLCMj



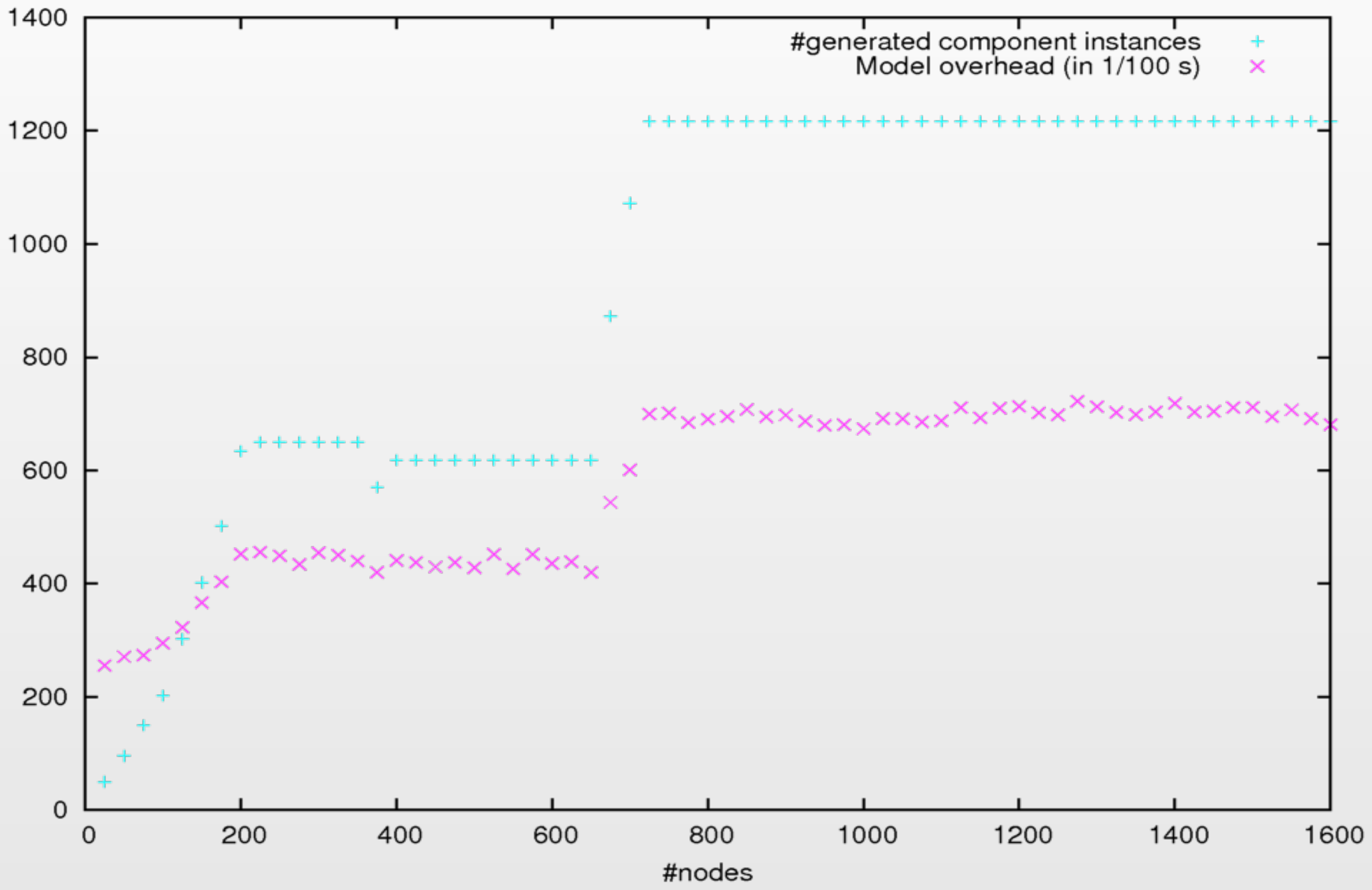
- Straitforward implementation of HLCM/Dhico & an heuristic to determine free parameters

HLCM: Hierarchical Programming Model

- Preliminary scalability experiment



HLCM: Hierarchical Programming Model



Conclusion

Current Status & Ongoing Work

- HLCM
  - Component, genericity, hierarchy, connector, open connection, component&connector implementation choice
  - Static model
    - Dynamicity to be added
  - HLCMi, an operational implementation
- HLCM/Charm++
  - Dedicated language for describing primitive component
  - Parser operational
  - Dumper / Launcher to be done
- OpenAtom & HLCM/Charm++
  - Synthetic version of PairCalculator/Ortho to be developed
    - Model & tool validation
  - Real experiments with PairCalculator/Ortho to be done