Towards MapReduce Skeleton in High Level Component Model

Christian Perez
Graal/Avalon INRIA EPI
LIP, ENS Lyon, France

Kickoff Meeting, ANR MapReduce
Rennes, 29-30 November 2010

Outline of the talk

- Motivation
  - MapReduce
- Overview of HLCM core concepts
- ANR MapReduce & HLCM
  - Towards MapReduce Skeletons
  - Subtask 5.1
  - Deliverables
  - Collaborations?
- Conclusion

Objectives

- Enable code-reuse
  - E.g. mapper or reducer code
  - Let expert develop a piece of code not tied to a framework
- Enable *adaptation* when re-using code
  - E.g. reducer “sum” not specific to a particular type of data
  - Let re-use code with parameterization options
- Enable any kind of composition operators
  - E.g. mapper or reducer may interact with a DB
  - Do not impose any communication models (framework)
- Enable efficient implementation of composition operators
  - E.g. enable resource specific optimization

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)

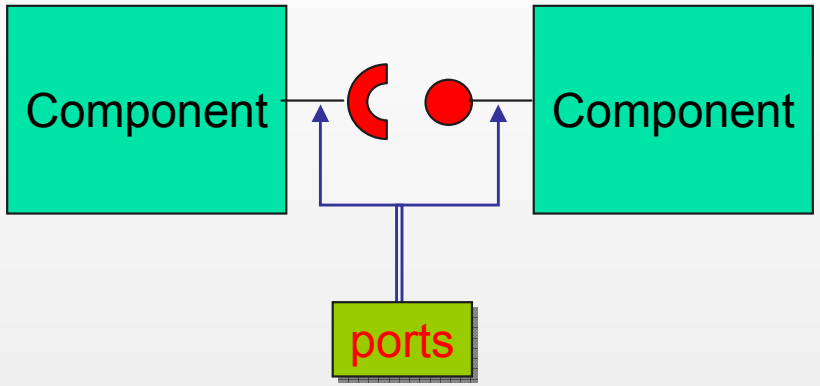
Component, Connector, Hierarchy,
Genericity, & Template Meta-Programming

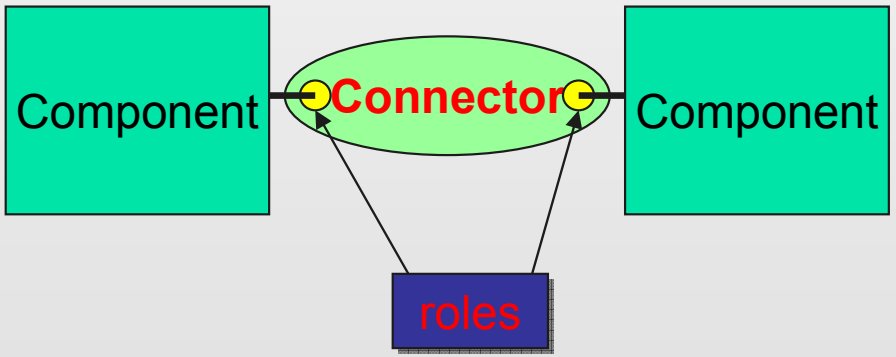
HLCM: High Level Component Model

- 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*

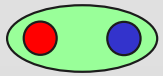
Connectors

- 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



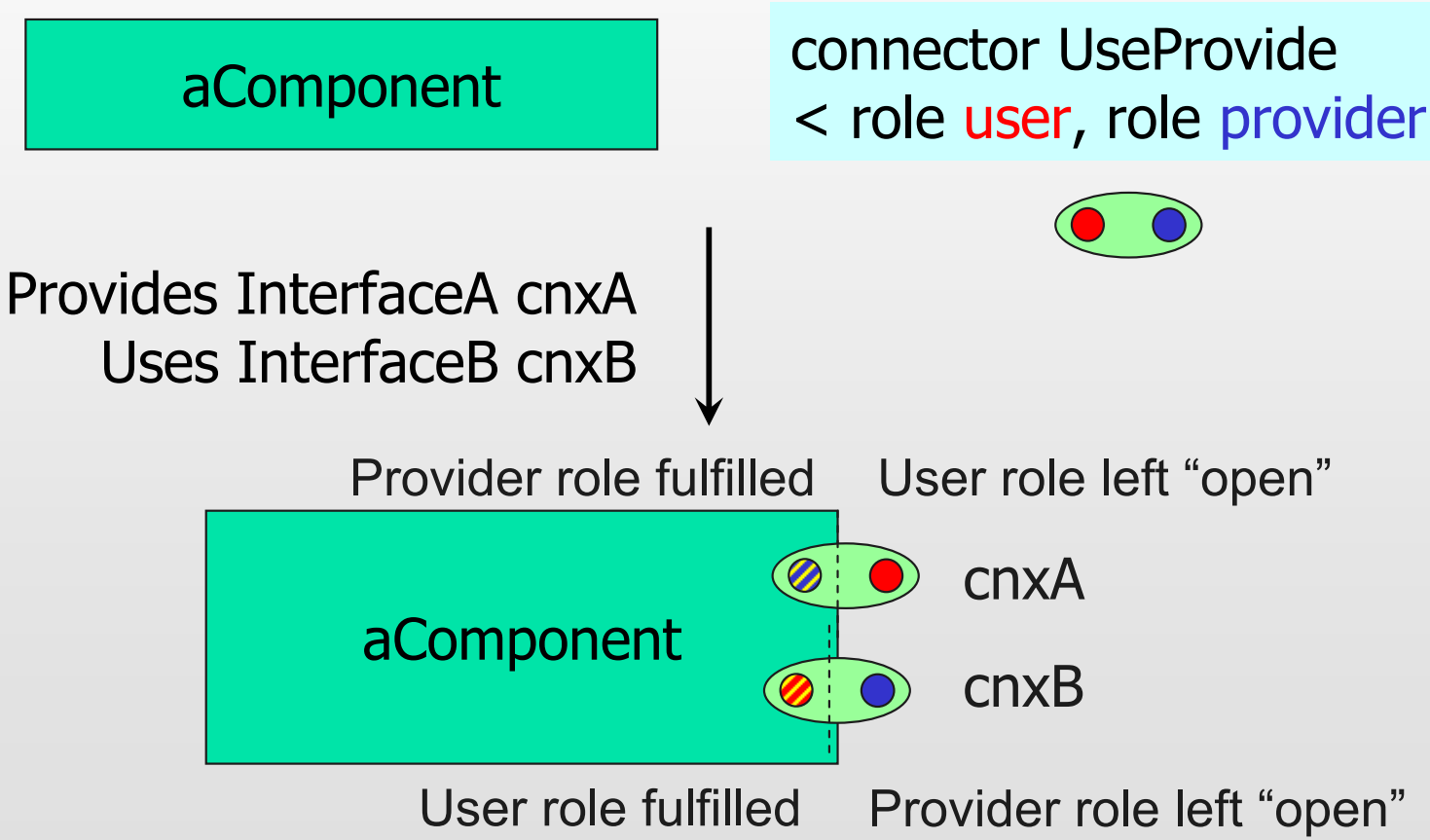


connector UseProvide
< role user, role provider >;



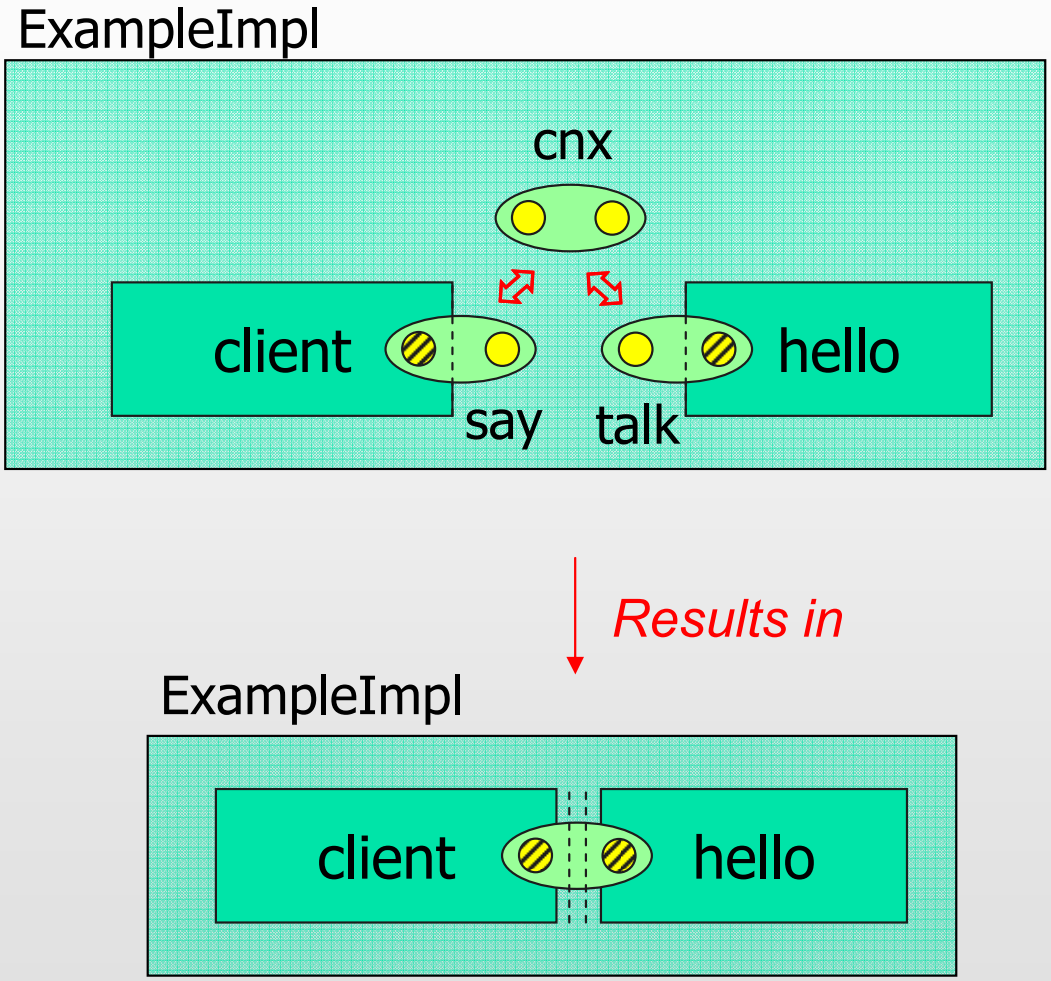
HLCM: Component & Connector

- Black box that may expose some open connections



HLCM: Composite Component

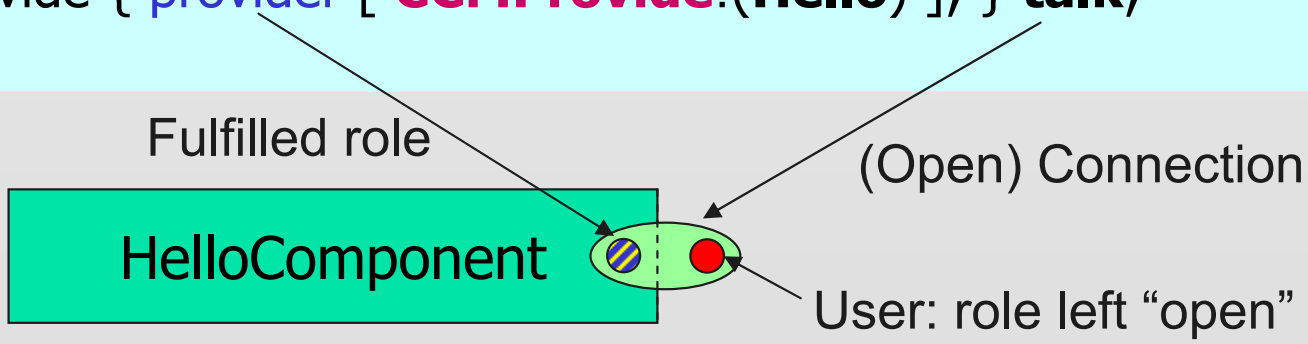
```
component  Example { }

composite  ExampleImpl
implements  Example
{
   HelloComponent hello;
   ClientComponent client;

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



HLCM: Primitive Components

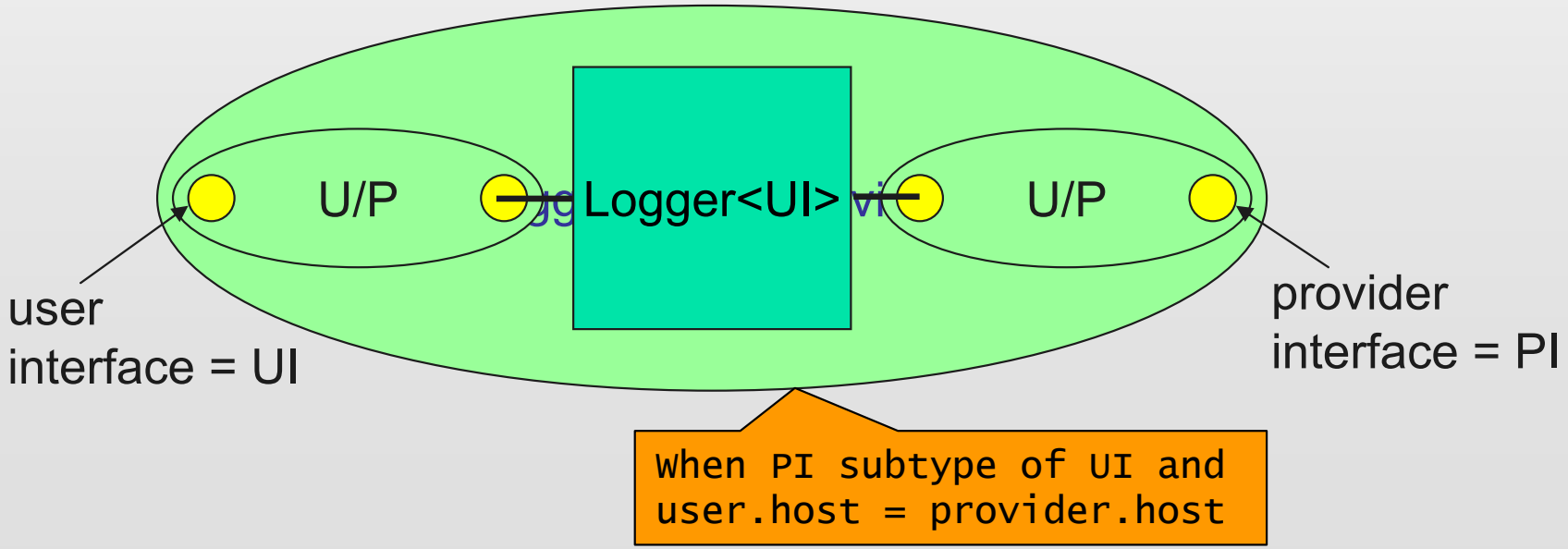
- Abstract Component Model
  - Primitive components not defined directly by HLCM
  - Primitives defined by a specialization
    - HLCM/Java, HLCM/C++, HLCM/CCM
- HLCM/CCM
  - Primitive component: CCM component
  - Primitive connector: UseProvide interactions

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

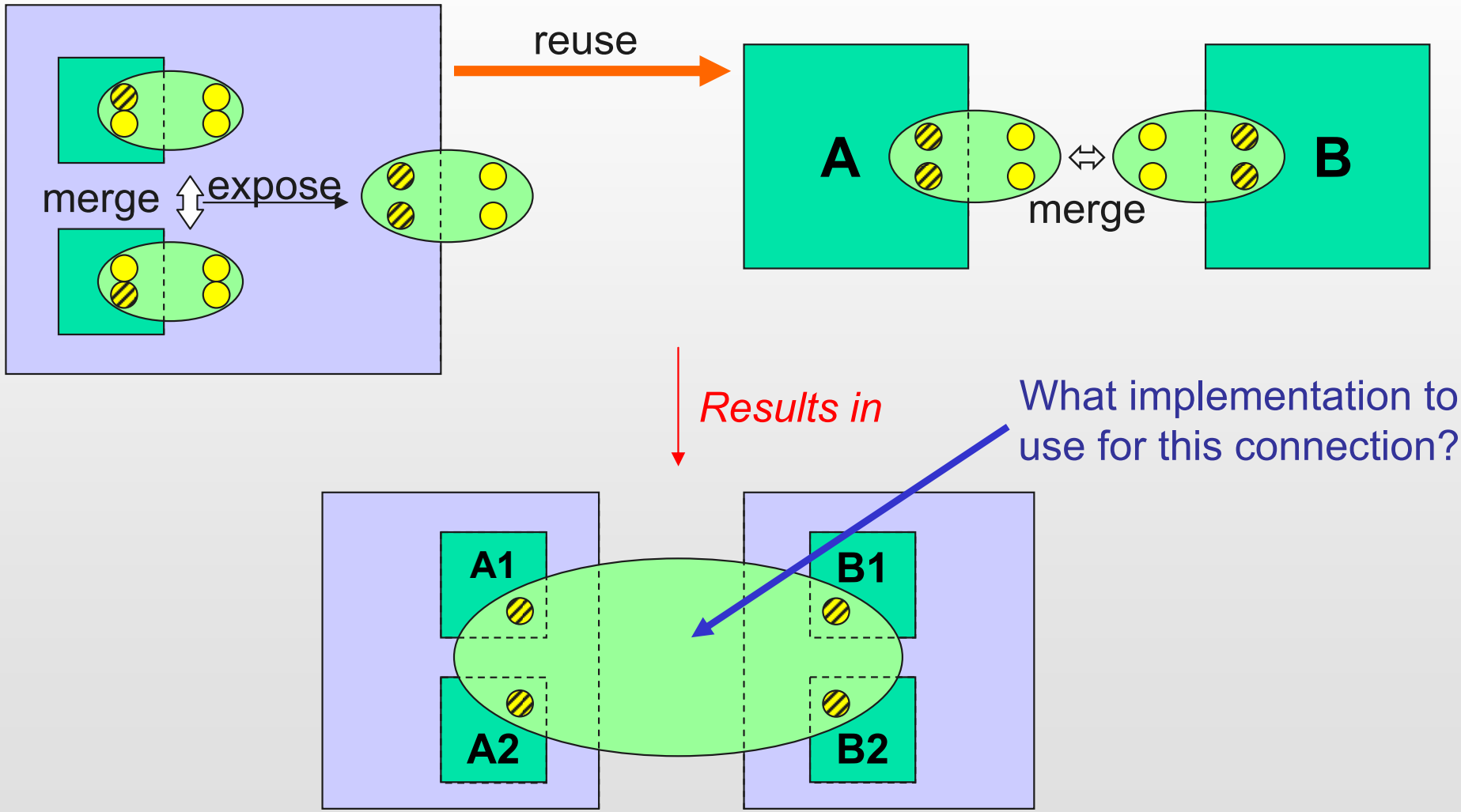


HLCM: User Implemented Connector

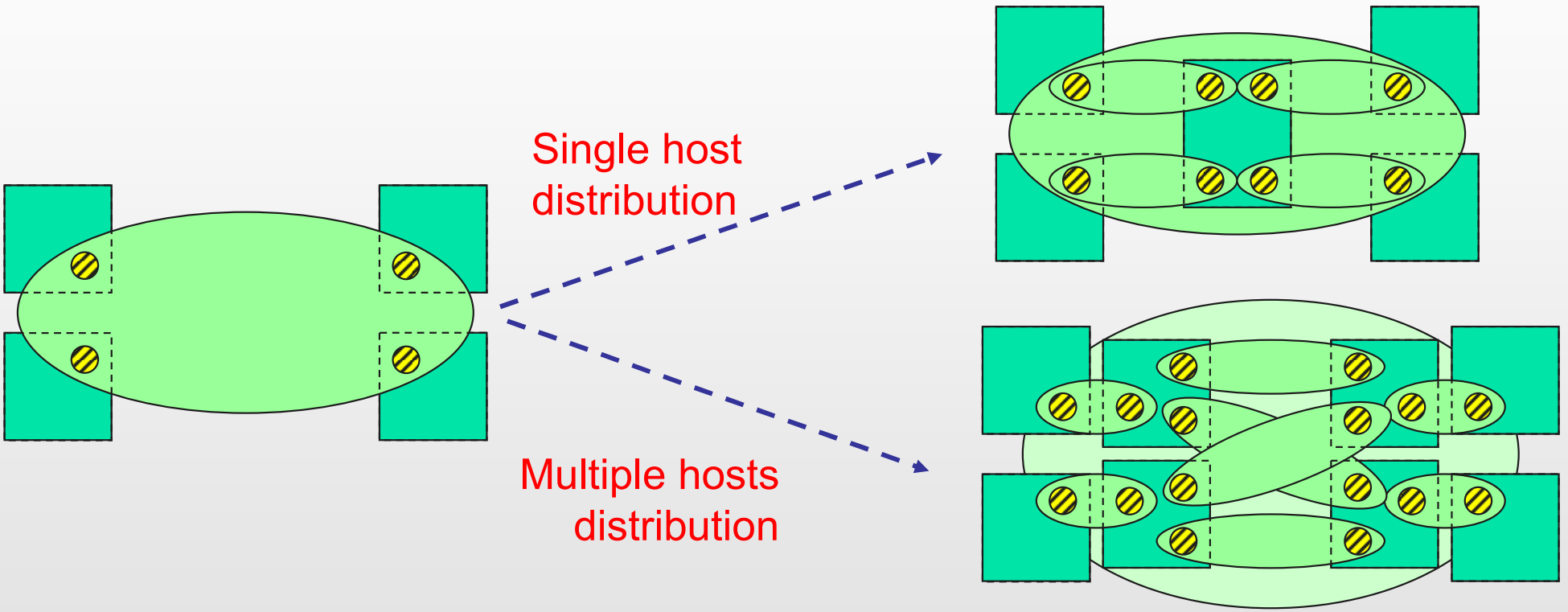
```
generator    LoggingUP<UI,PI>
Implements   UseProvide<provider = { CCMProvide!<PI> },
                        user     = { CCMUse!<UI>     }>
when ( UI super PI )
{
   Logger<UI> proxy;
   proxy.clientSide.user += this.user;
   proxy.serverSide.provider += this.provider;
}
```



HLCM: Benefit of Open Connections



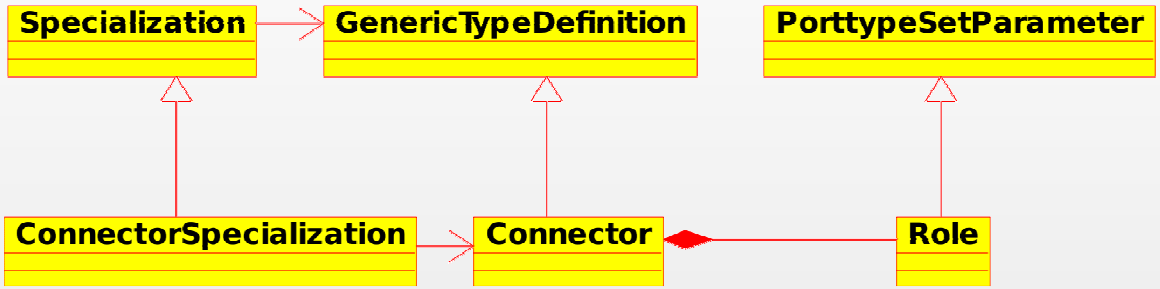
HLCM Connection Implementation: a Planning Choice



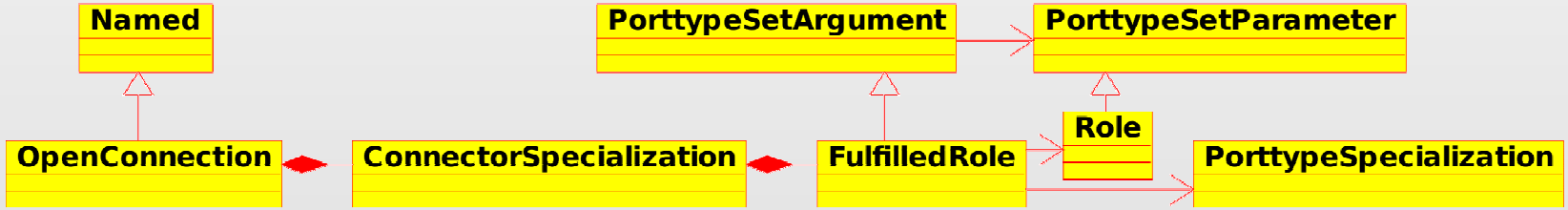
- Component and connection implementation choice made by *choosers*
  - Not defined in HLCM
  - Specialization depend

Model based HLCM Definition

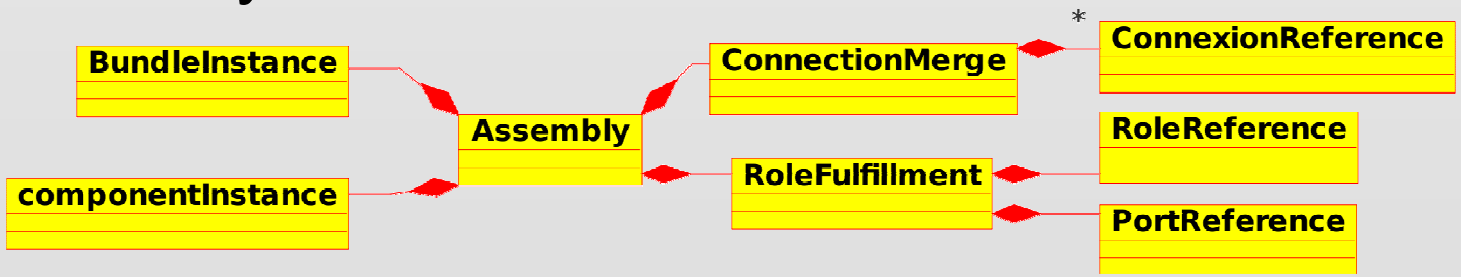
- Connector



- Connection



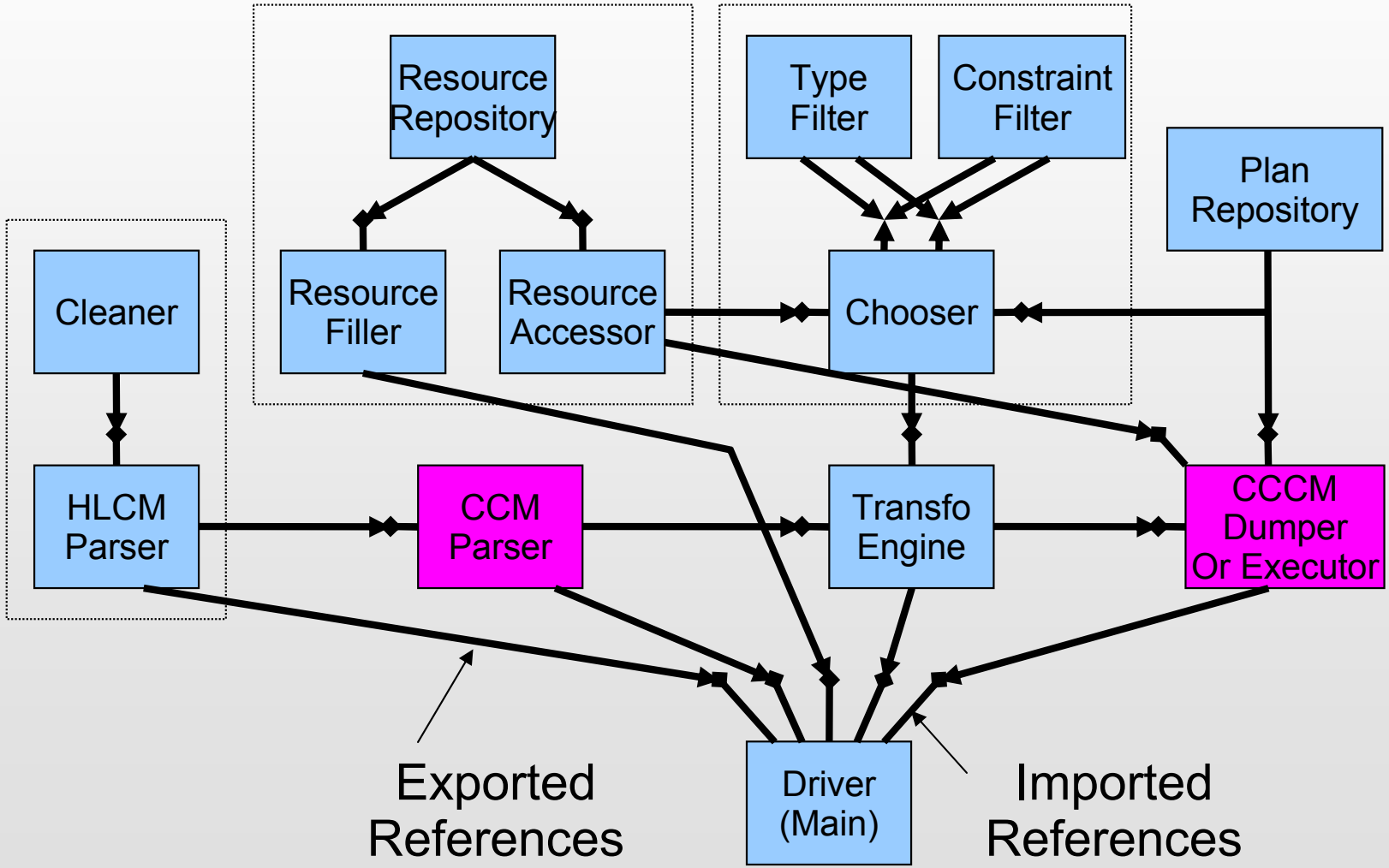
- Assembly



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/CCM in LLCMj



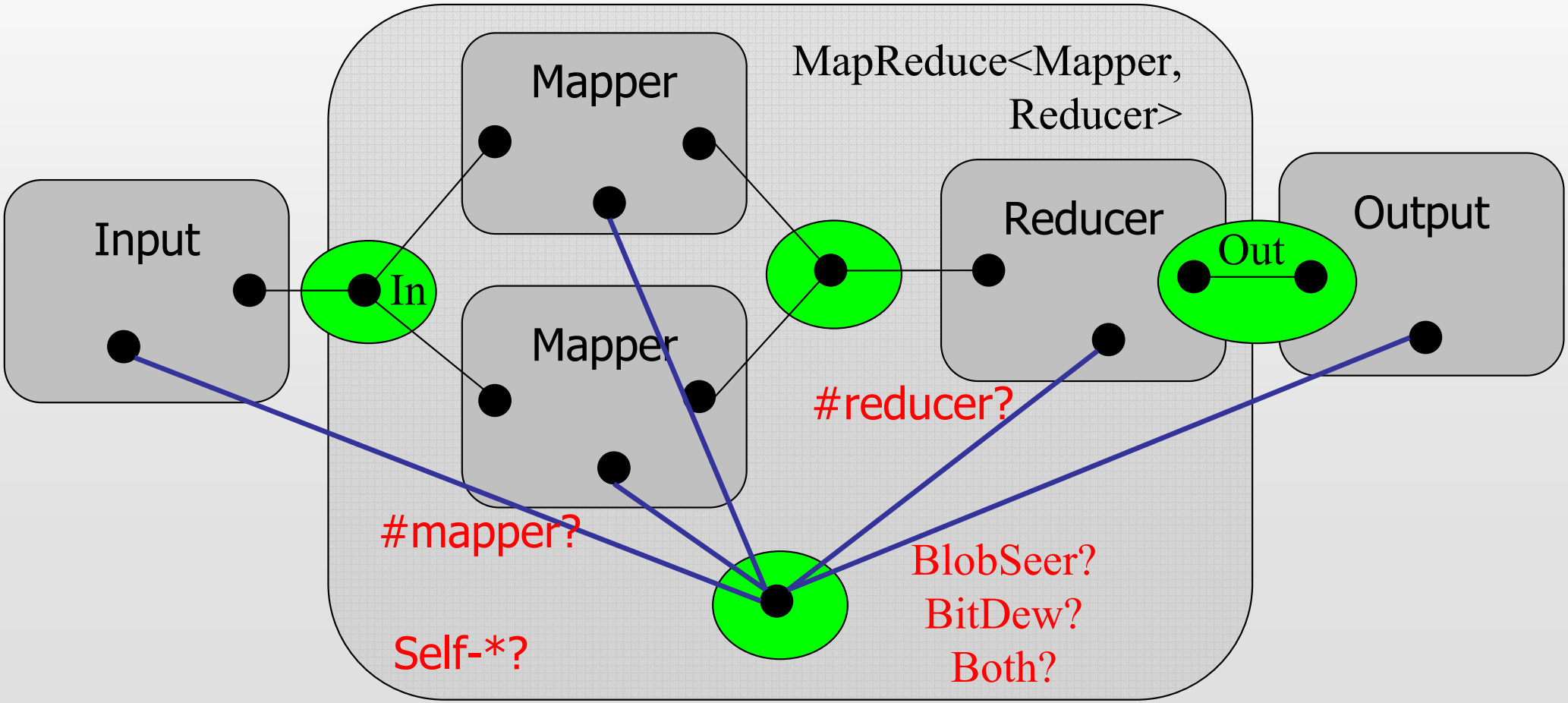
ANR MapReduce & HLCM

Subtask 5.1

Application programming and deployment

Towards a MapReduce Skeleton in HLCM

Component MapReduce<Component Map, Component Reduce>
exposes { In, Out}



Subtask 5.1: Three Stages

- A coherent and easy-to-use programming model
  - *Check whether HLCM is ok*
- Generate an executable (i.e., deployable) application
  - Depend of the requirements of Task 5.
  - *Integrate the middleware layers resulting from Task 2, Task 3.1 and Task 3.2 into HLCM*
    - Define & implement adequate connectors
- Provide a tool to deploy the resulting executable to targeted infrastructures (G5K, FutureGrid, etc)
  - *Adapt ADAGE, or integrate HLCM&Adage, or ?*
    - Depend on requirements
    - Support elastic resource

Deliverables

- T0+18 [D5.1] (Stage 1 & 2)
  - A study on the definition of Map-Reduce skeletons into a template-based general purpose component model and the usage of model transformation techniques to automatically integrate middleware artifacts into a deployable application (report).
- T0+24 [D5.2] (Stage 3)
  - A study on the adaptation of the generic deployment tool Adage to clouds in general, and Nimbus in particular (report).
- T0+36 [D5.3]: A set of integrated prototypes (software).

Subtask 5.1: Identified Collaborations

- Kerdata
  - Integration Blobseer & HLCM
- Graal
  - Integration Bitdew & HLCM
  - Integration of scheduling algo into HLCM
- JointLab INRIA-UIUC Lab
  - FT & HLCM to be discussed
- IBCP/MEDIT SA
  - Bioinformatics application & HLCM

Conclusion

- HLCM
  - Component, genericity, hierarchy, connector, open connection, component&connector implementation choice
  - Static model
    - Dynamicity to be added
  - HLCMi, an operational implementation

- Open Questions
  - Do we need dynamicity support?
  - Which primitive component model(s)?