On High Performance Software Component Models

Christian Perez
Christian.Perez@inria.fr
GRAAL/AVALON INRIA Project-Team
LIP, ENS Lyon, France

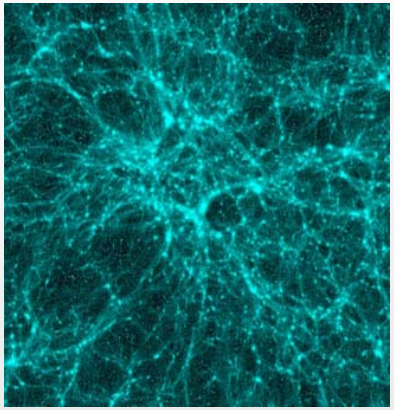
HPC, Grids and Clouds Workshop
Cetraro, Italy, 25 June 2010

Outline of the talks

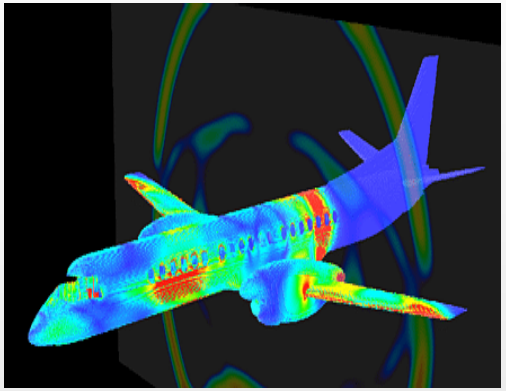
- Overview of component models
- Examples of composition operators through applications
  - MxN
  - Master-Worker
  - Data-sharing
  - Spatial & temporal composition
  - Algorithmic skeletons
- Towards a concept-extensible component model
  - HLCM
- Conclusion

Applications: Numerical Scientific Simulations

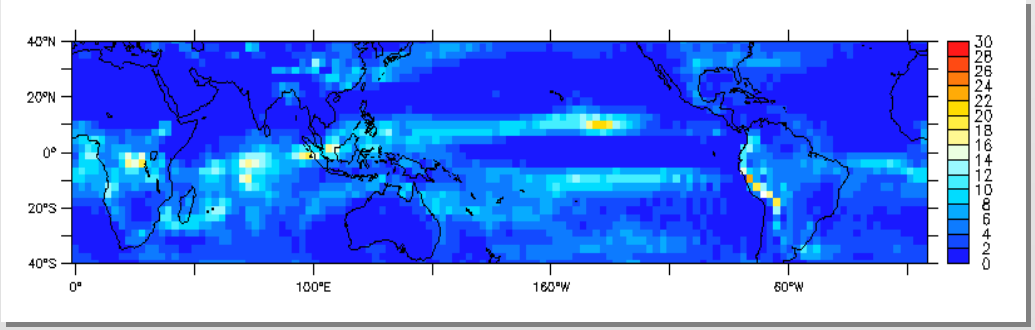
- More and more complex
  - Code coupling
- Need model to handle such complexity
  - Simple
  - Efficient



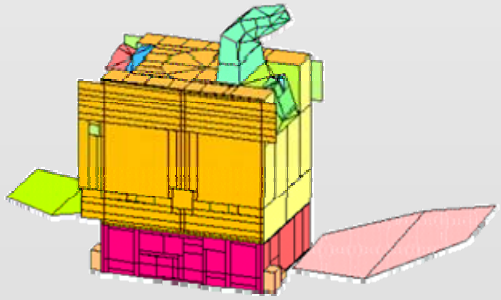
Astrophysics



Electromagnetism



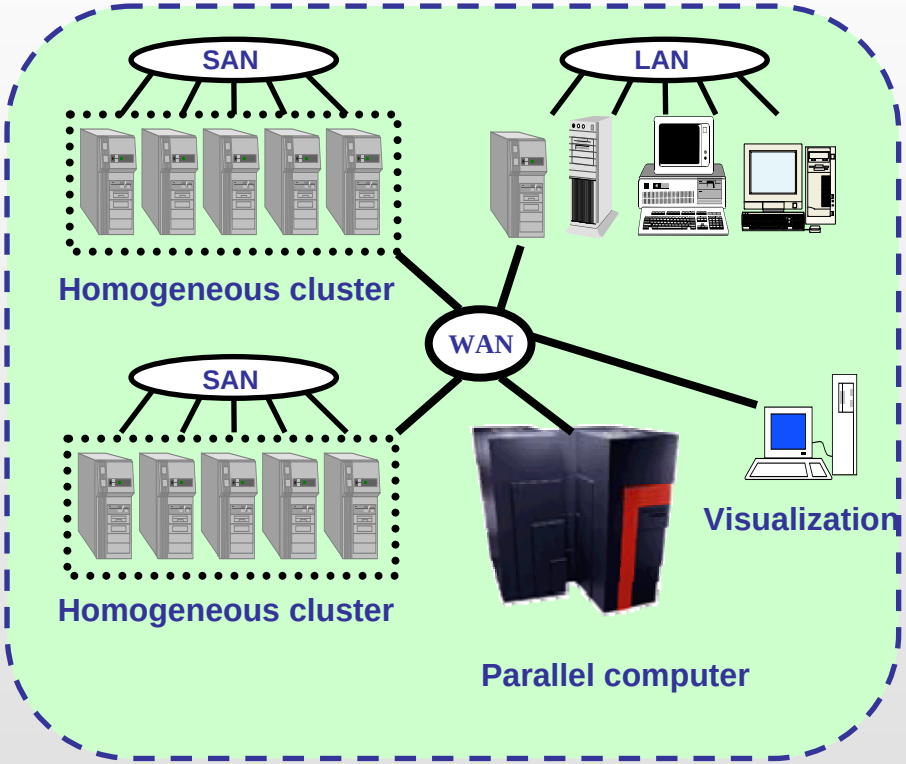
Climatology



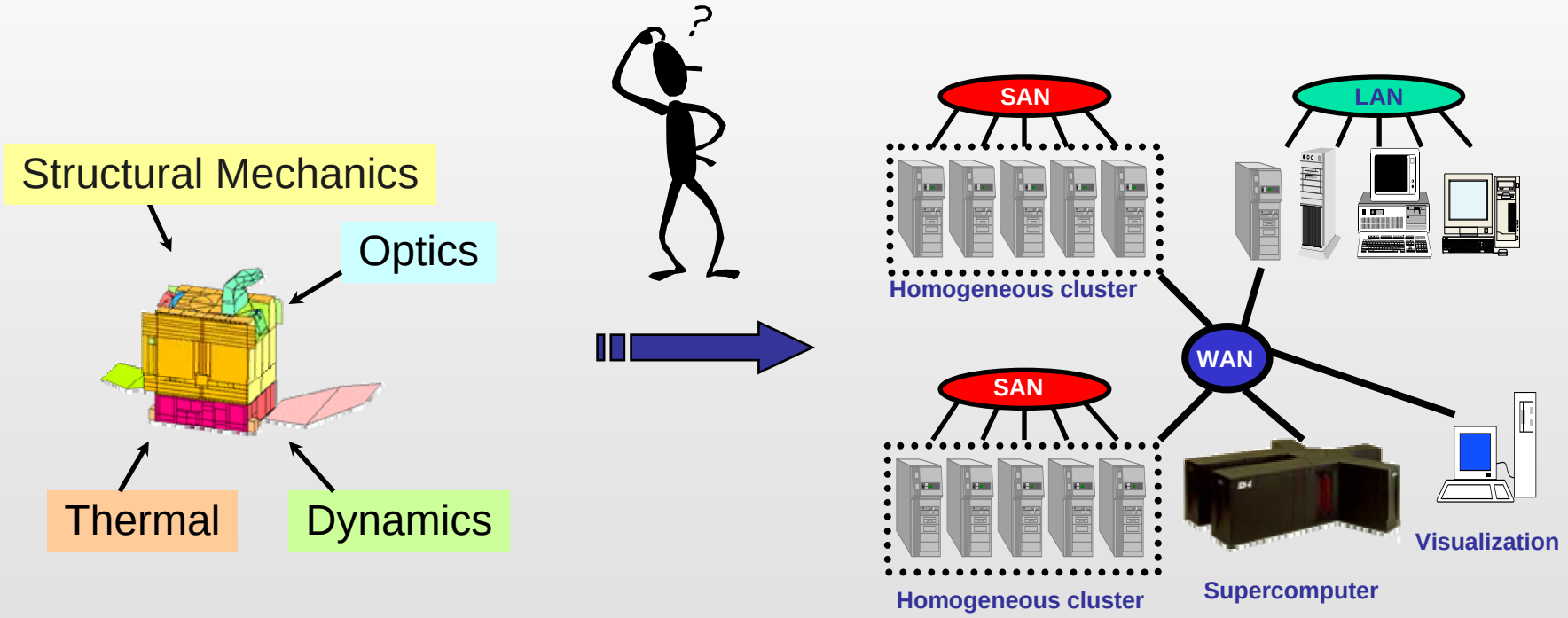
Satellite

The Super/Grid/Cloud/Sky Computer

- Hierarchical networks
  - WAN
    - Internet, Private WAN, etc.
  - LAN
    - Ethernet
  - SAN
    - Infiniband, …
- Computing resources
  - Homogeneous clusters
    - Many-core nodes
    - With/without GPU
  - Super computers
  - …
- Fast evolution!
- Heterogeneity!

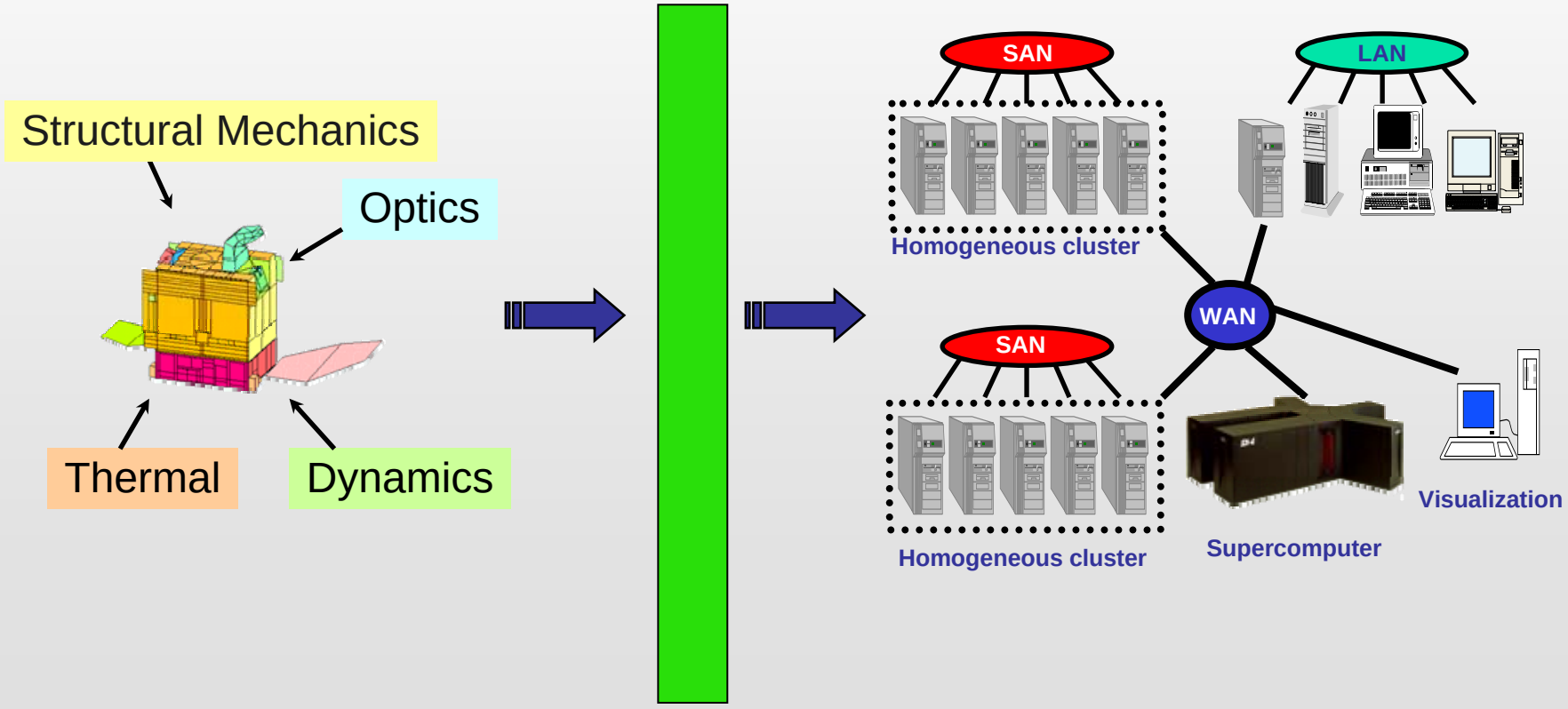


Application on Resources?



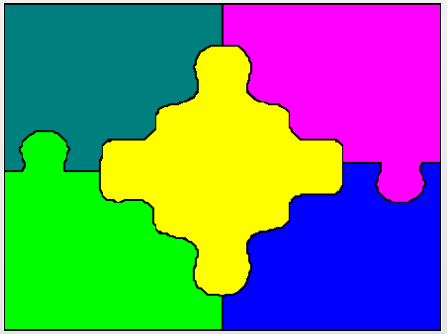
Application on Resources? Abstractions!

Programming Environment



Software Component

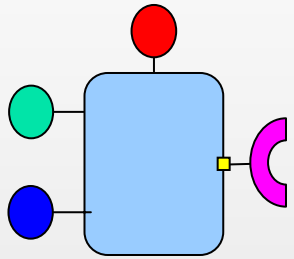
- Technology that advocates for composition
  - Old idea (late 60's)
  - *Assembling* rather than *developing*
- Aim of a component: to be composed!
  - Spatial composition
  - Temporal composition



- Example of success stories
  - Pipe (in OS)
  - Data flow models

Software Component: Black Box and Ports

- A component is a black box that interacts by its ports
- Port ~ access point
  - Name
  - Description and protocol
- Usually
  - (Object) Interface
  - Message passing

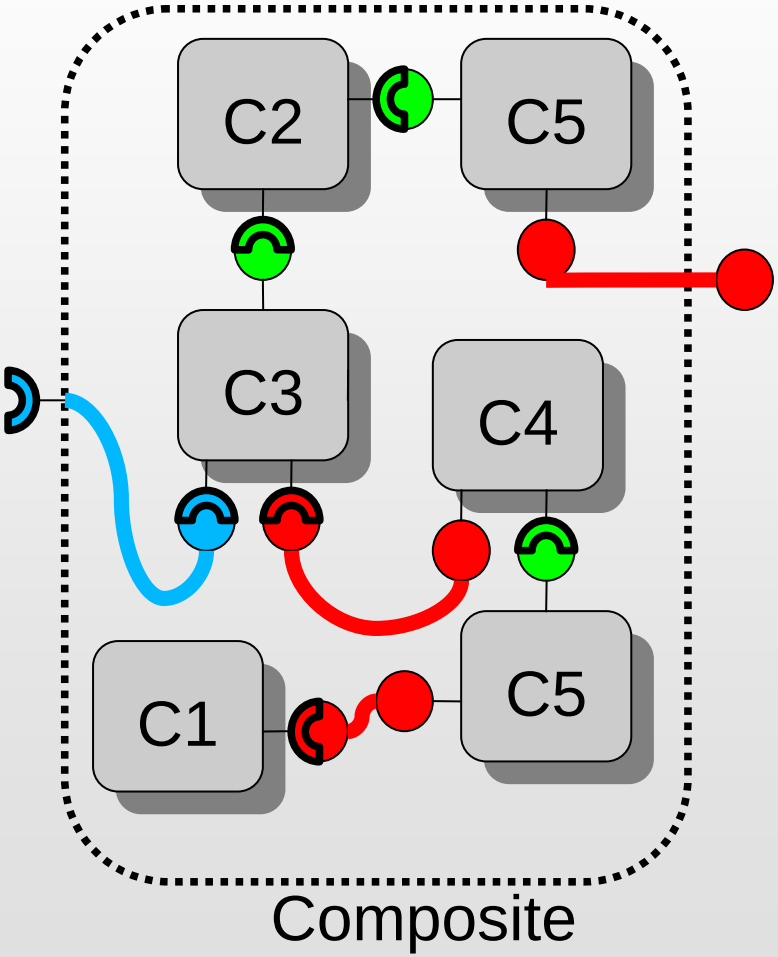


```
component MyComponent
{
    provides IExample1  to_client1;
    provides IExample2  to_client2;
    uses     Itfaces2   to_server;
};
```

```
interface IExample1
{
    void factorise(in Matrix mat);
};

interface IExample2 {  …  };

interface Itfaces2 {  …  };
```

Assembly of Software Components

- Description of an assembly
  - Dynamically (API)
  - Statically
- Architecture Description Language (ADL)
  - Describes
    - Component instances
    - Port connection
  - Available in many CM
- Primitive and composite components



Many component models

- Industry (mainly distributed models)
  - EJB – Entreprise Java Bean (SUN)
  - CCM – CORBA Component Model (OMG)
  - SCA – Services Component Architecture (OSOA)
  - .NET (Microsoft)
  - …
- Research
  - CCA – Common Component Architecture (CCA Forum)
  - Fractal (France télécom & INRIA)
  - GCM – Grid Component Model (CoreGrid)
  - Grid.it / ASSIST (UNIPI)
  - …

Which Usage for Components?

- Components as an execution model entity
  - Components are usually thought to describe the actual architecture of an application at execution
    - Architecture == static structure
    - Modifications on the responsibility of the application
- Components as a programming model entity
  - Does it make sense?
    - Components are already a first class citizen of ADL
    - If applications will be made of components, there is a clear need to have high level component assembly language

Let introduce an hydrogeology application

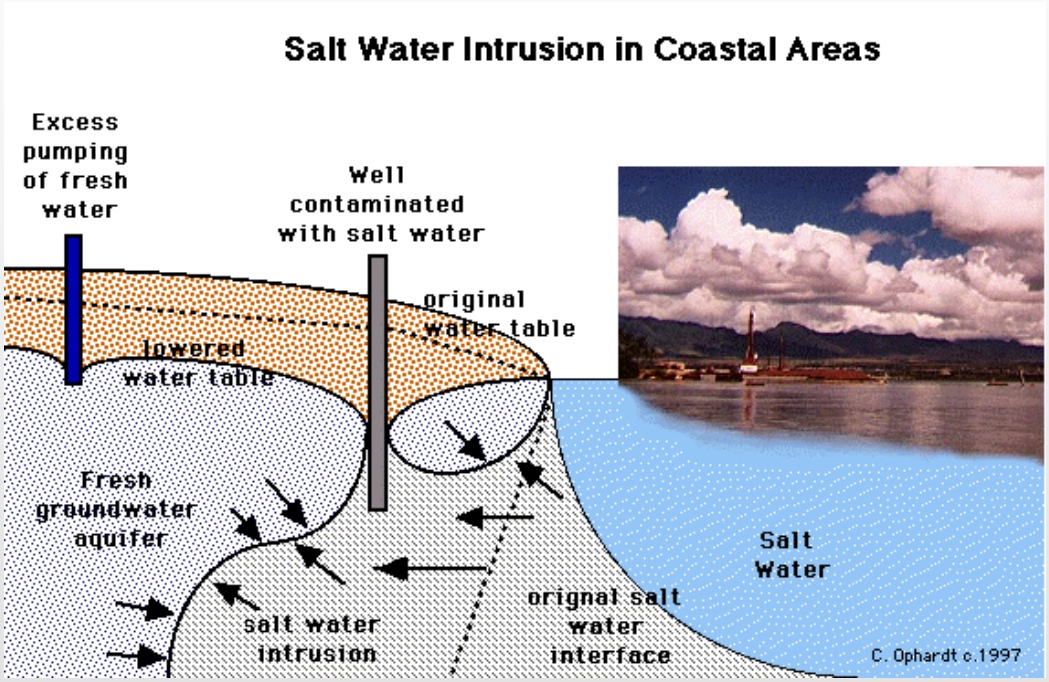
Parallel Component

MxN Communications

Application in Hydrogeology: Saltwater Intrusion

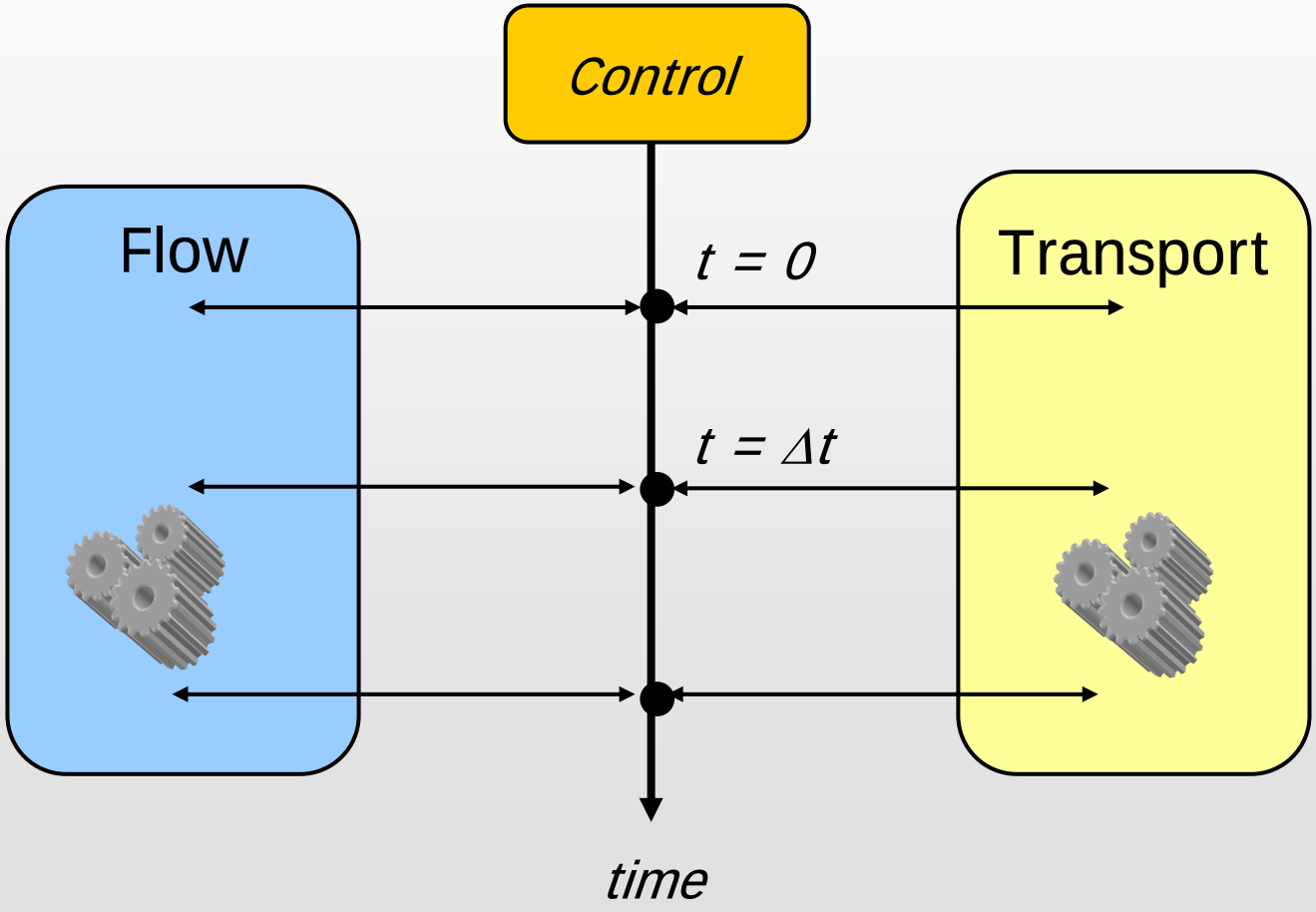
- Coupled physical models
- One model = one software
- Saltwater intrusion
  - Flow / transport
- Reactive transport
  - Transport / chemistry

- Hydrogrid project, supported by the French ACI-GRID



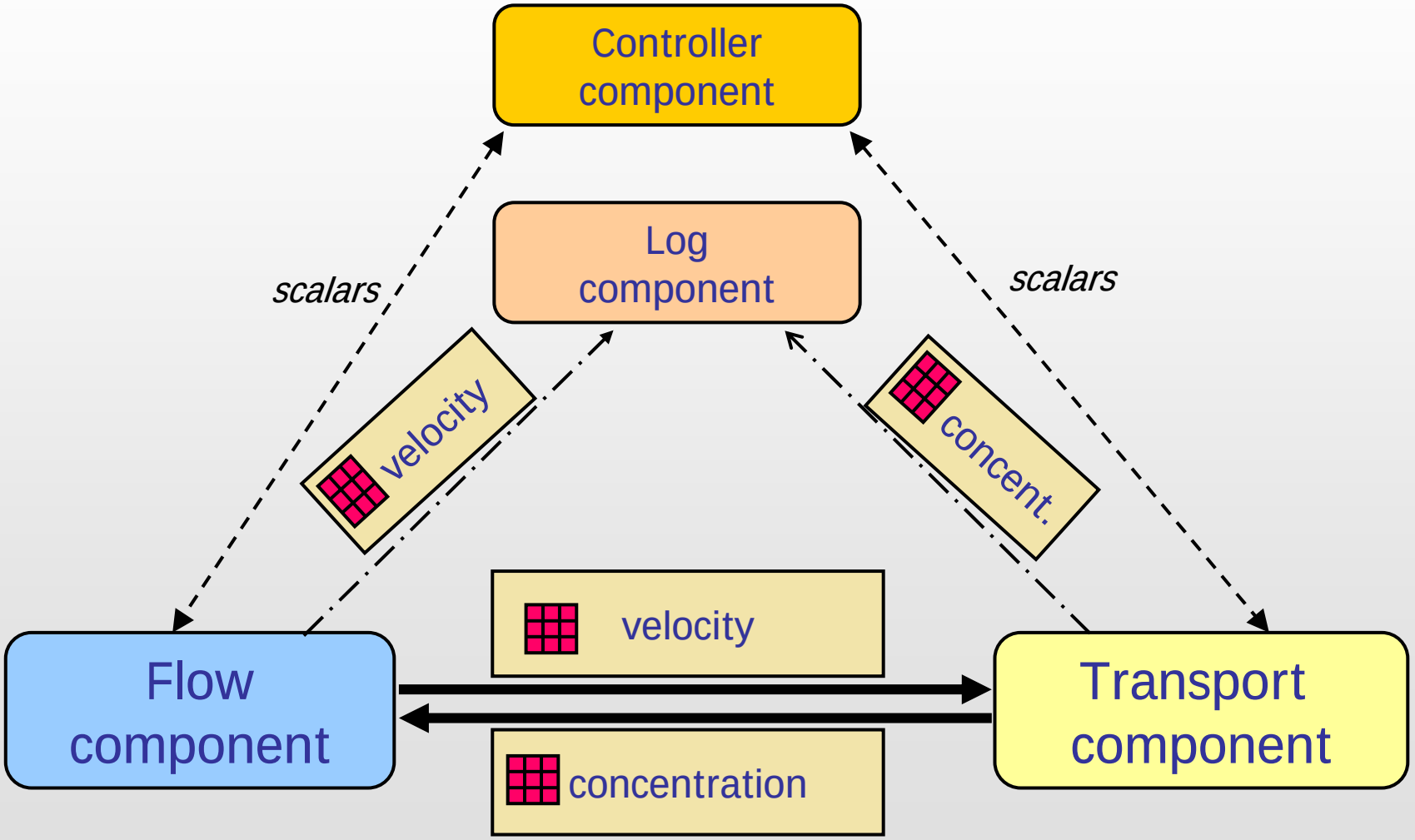
flow : velocity and pressure function of the density
Density function of salt concentration
Salt transport : by convection (velocity) and diffusion

Numerical coupling in saltwater intrusion

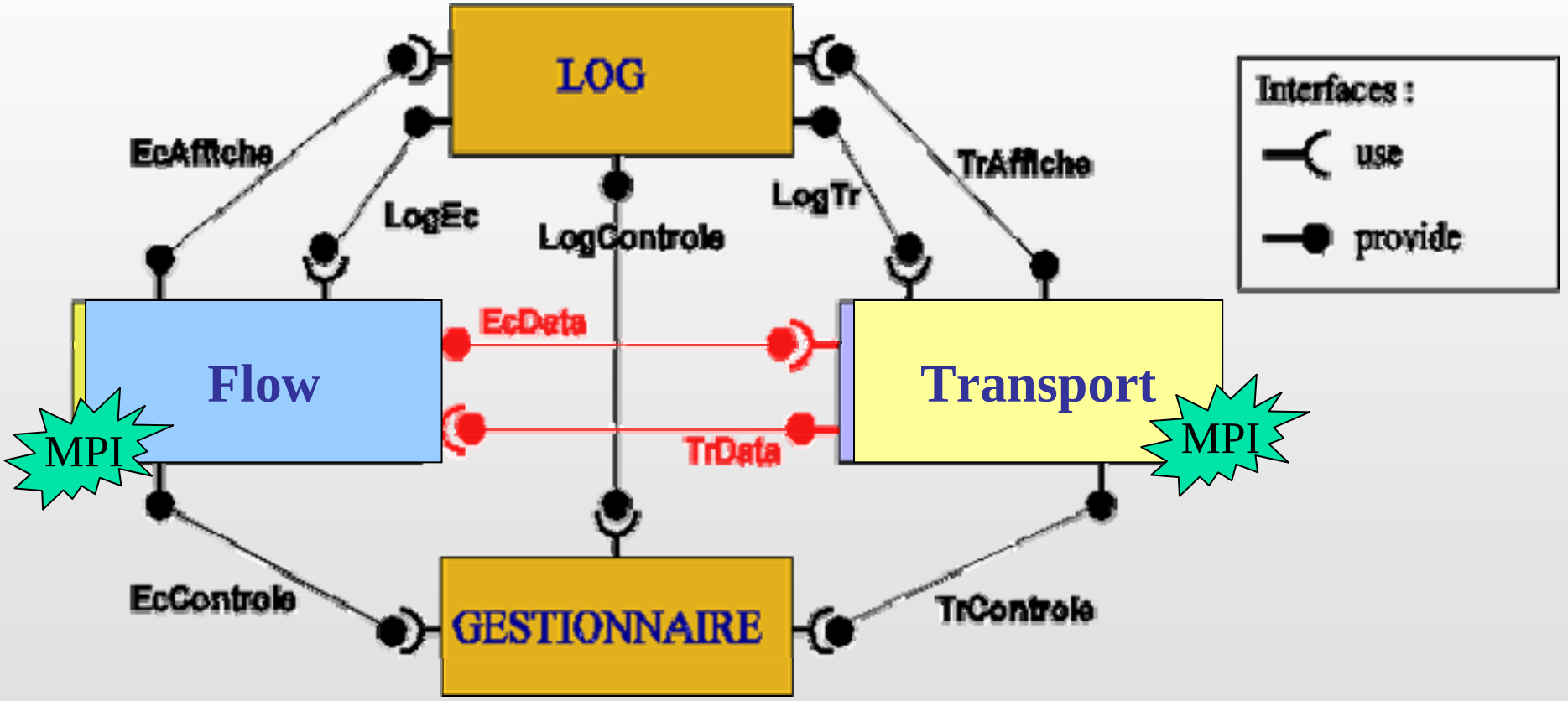


iterative scheme at each time step

Components and communications of PCSI

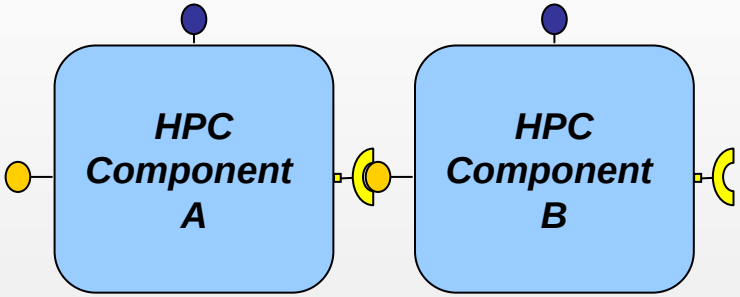


Components and interfaces of PCSI

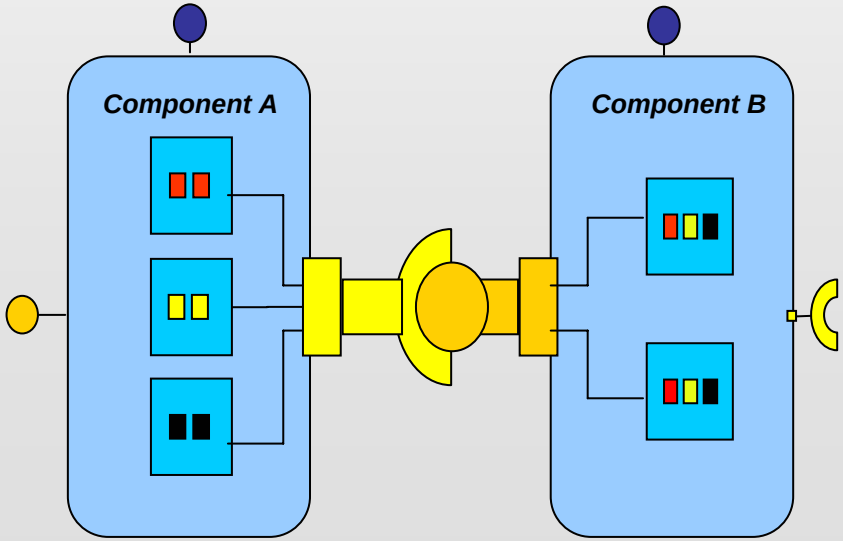


SPMD Components

What the application designer should see…

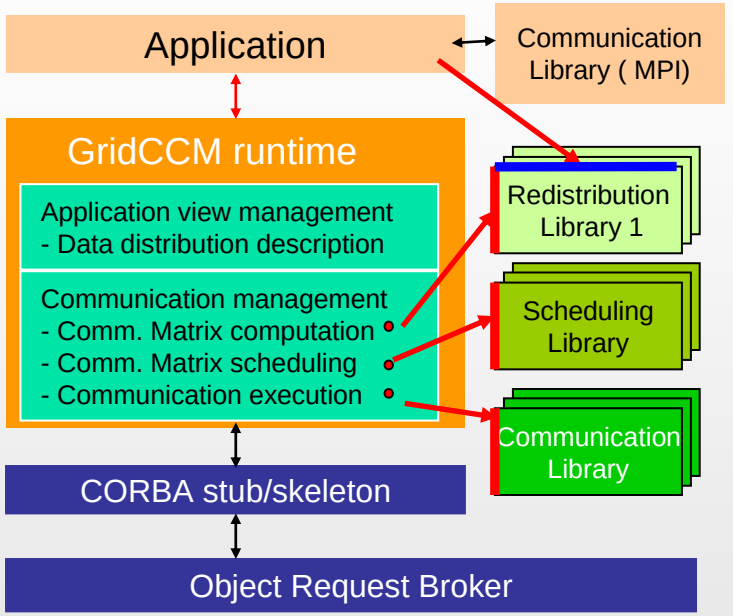


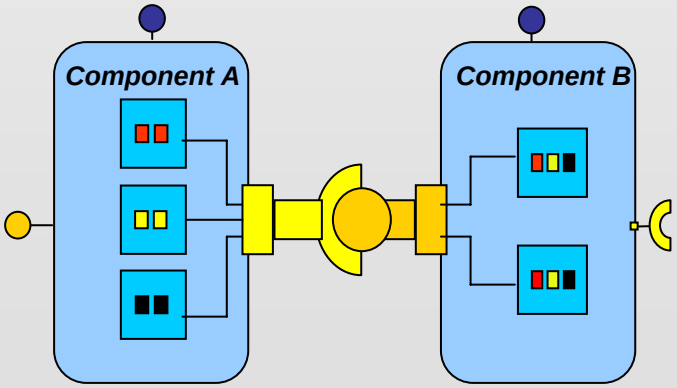
… and how it must be implemented !

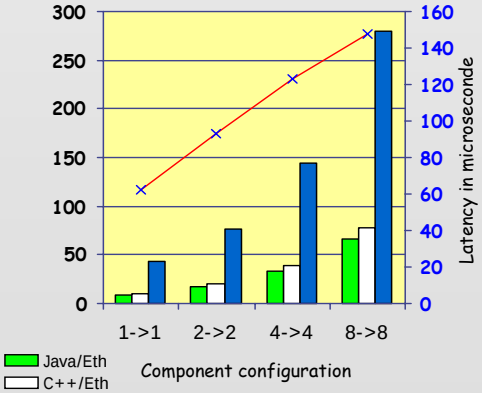


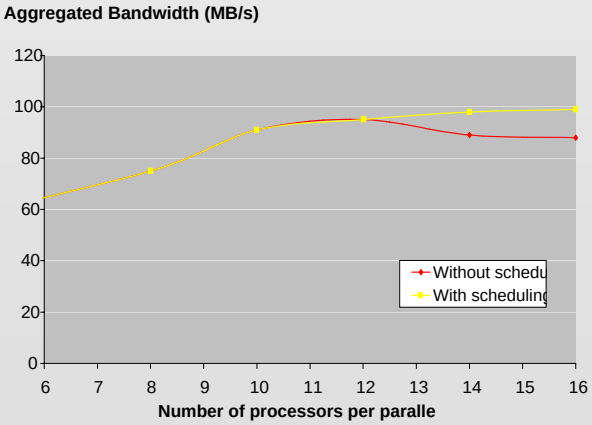
Components for code coupling: SPMD paradigm in GridCCM

- SPMD component
  - Parallelism is a non-functional property of a component
    - It is an implementation issue
  - Collection of sequential components
    - SPMD execution model
  - Support of distributed arguments
    - API for data redistribution
    - API for communication scheduling w.r.t. network properties
  - Support of parallel exceptions





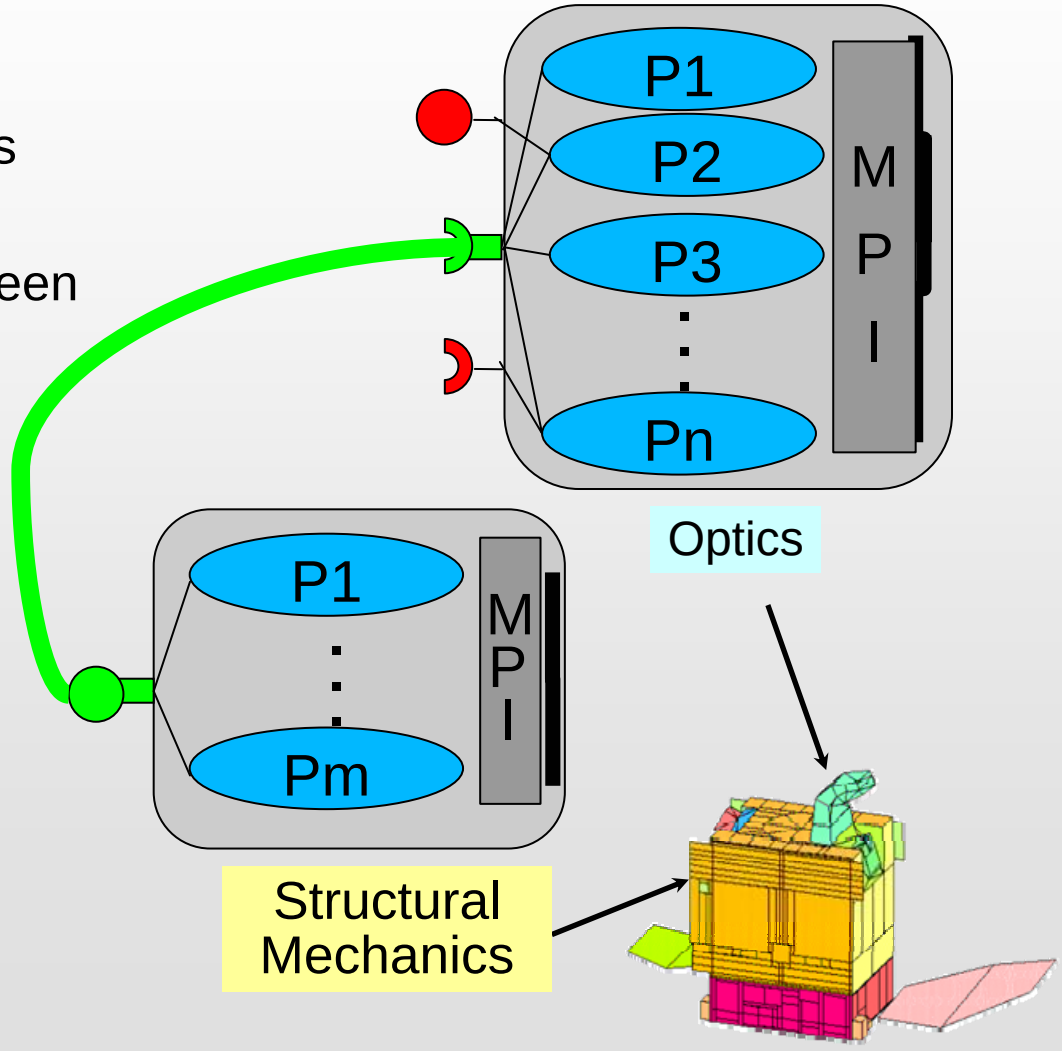




What about Collective Communications?

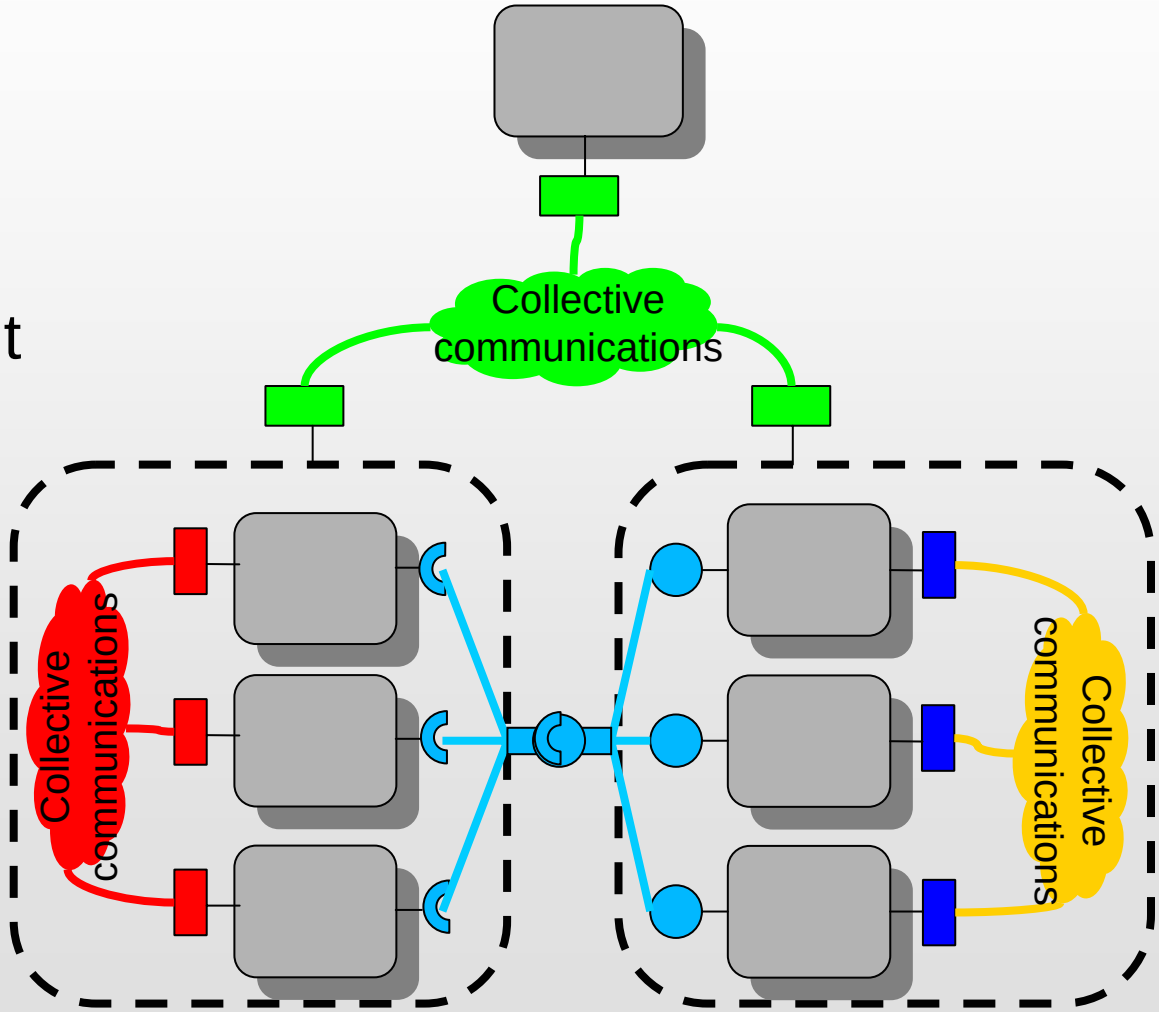
- Using message passing libraries (ex. MPI) inside components
- Using parallel ports (NxM) between components

- No collective communications at higher level
- Two communication models to handle



Collective Communications between Components

- Goals
  - Efficient
  - Transparent
  - Fits in component model
- Implementation
  - MPI
  - Point-2-Point comm.

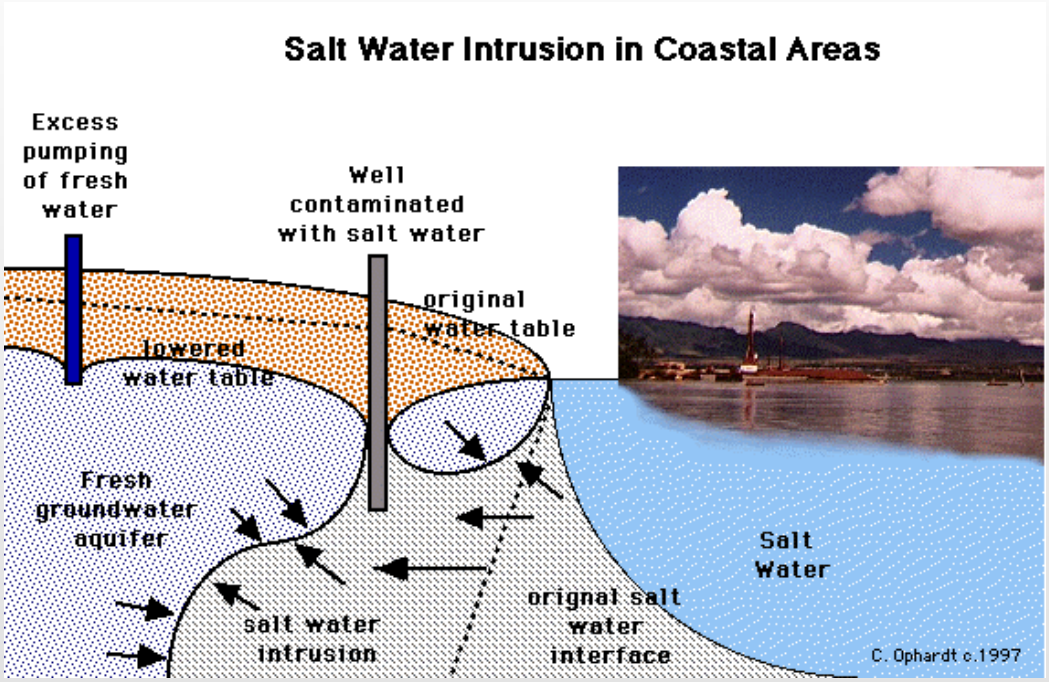


Let look at another hydrogeology application

Master-Worker Paradigm

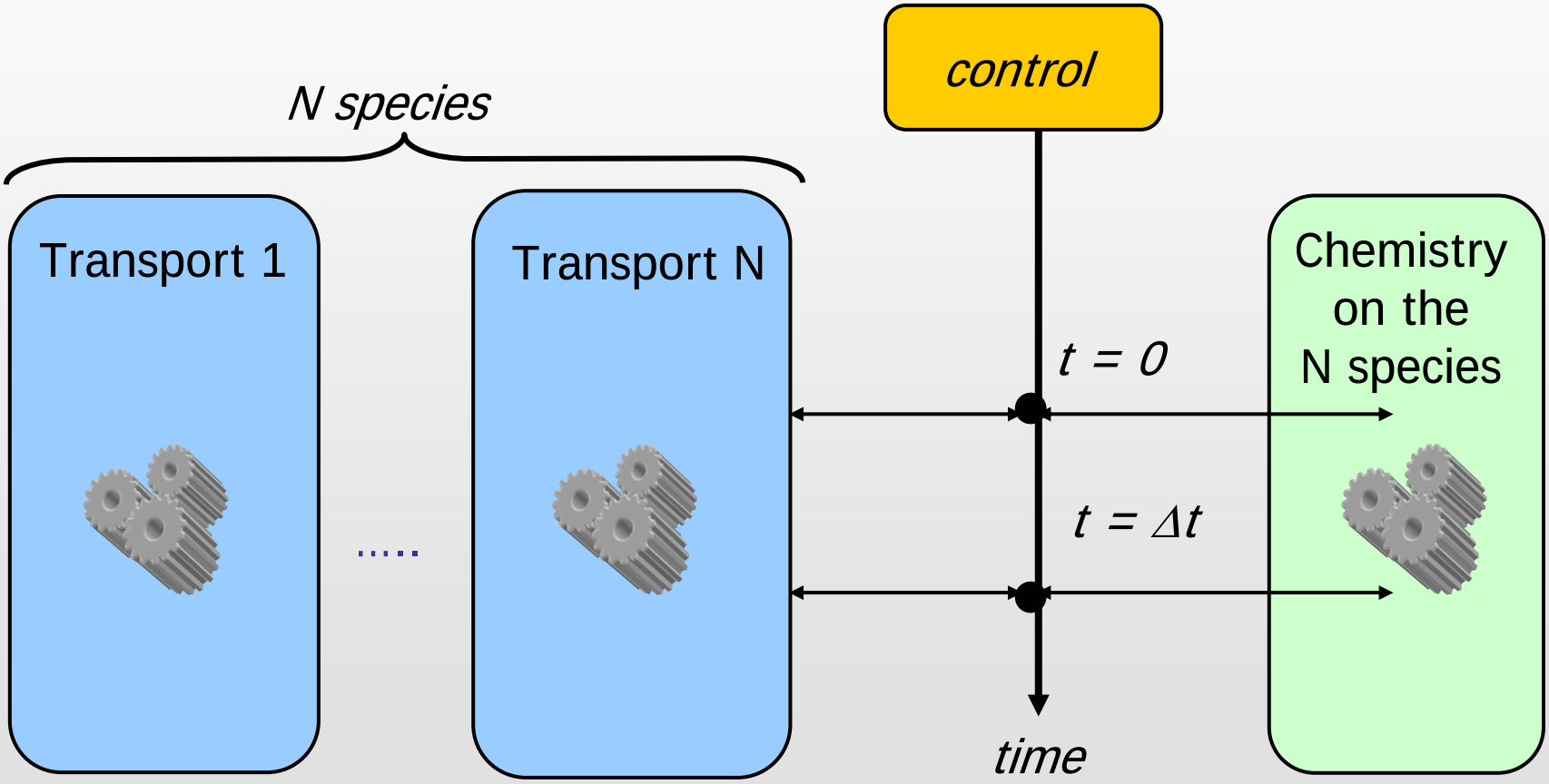
Application in Hydrogeology: Saltwater Intrusion

- Coupled physical models
- One model = one software
- Saltwater intrusion
  - Flow / transport
- Reactive transport
  - Transport / chemistry
- Hydrogrid project, supported by the French ACI-GRID



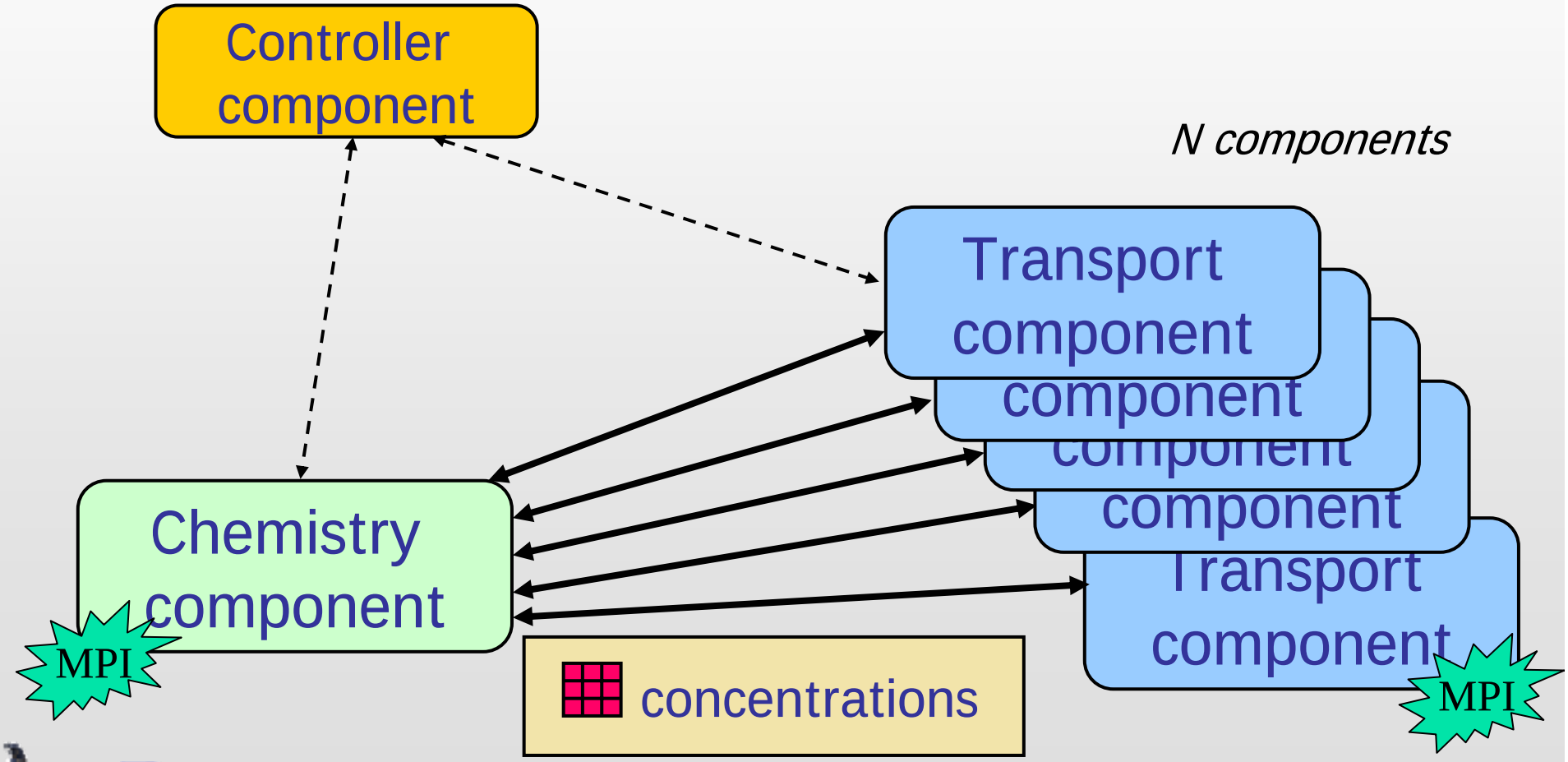
flow : velocity and pressure function of the density
Density function of salt concentration
Salt transport : by convection (velocity) and diffusion

Numerical coupling in reactive transport



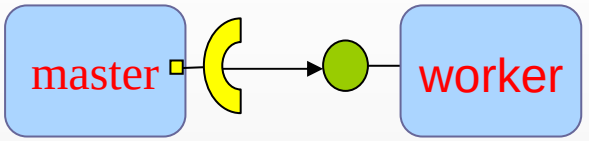
Iterative scheme at each time step

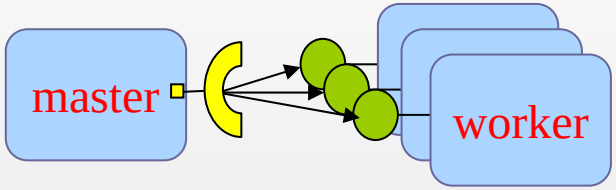
Component Model for Reactive Transport

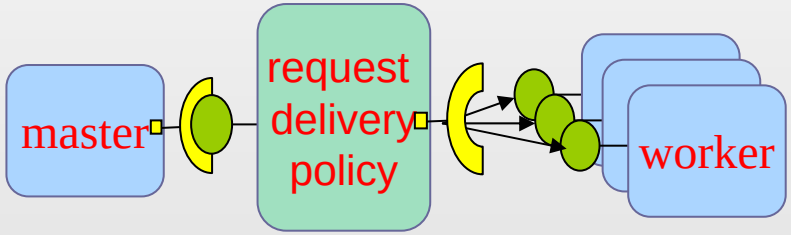


Limits with Existing Component Models

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



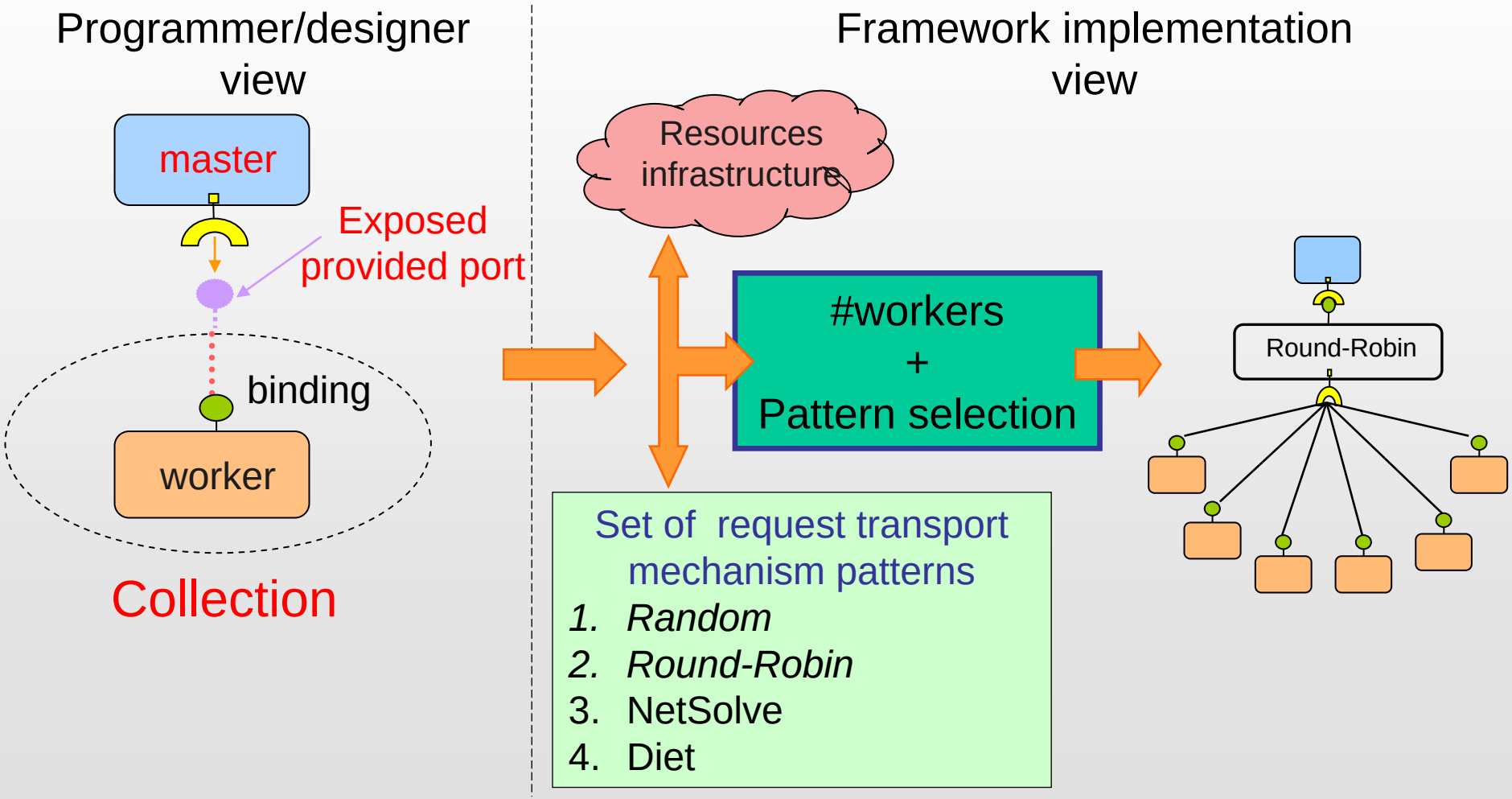




- At the burden of the programmer
  - Complex
  - No transparence

- Objectives
  - **Transparency**
  - **Re-use existing Master-Worker environments**

Master-Worker oriented Component Collection



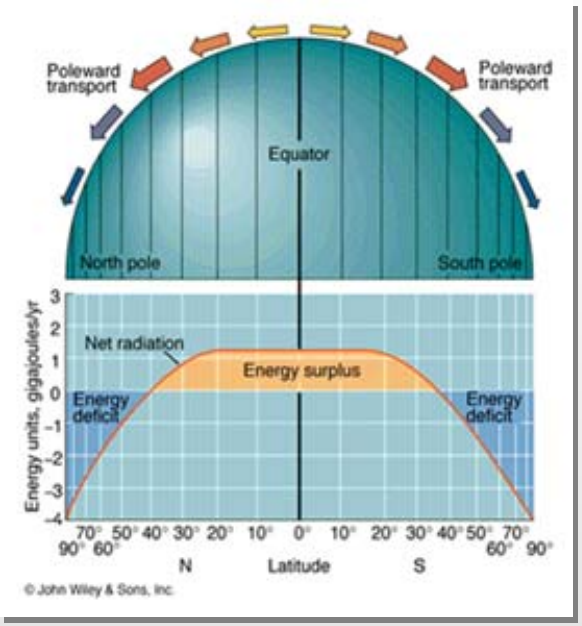
Let examine at a climatology application

Multiple composition operators

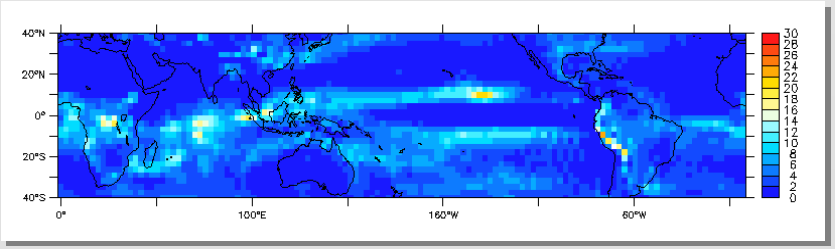
Composing middleware

Ocean-Atmosphere Numerical Simulations

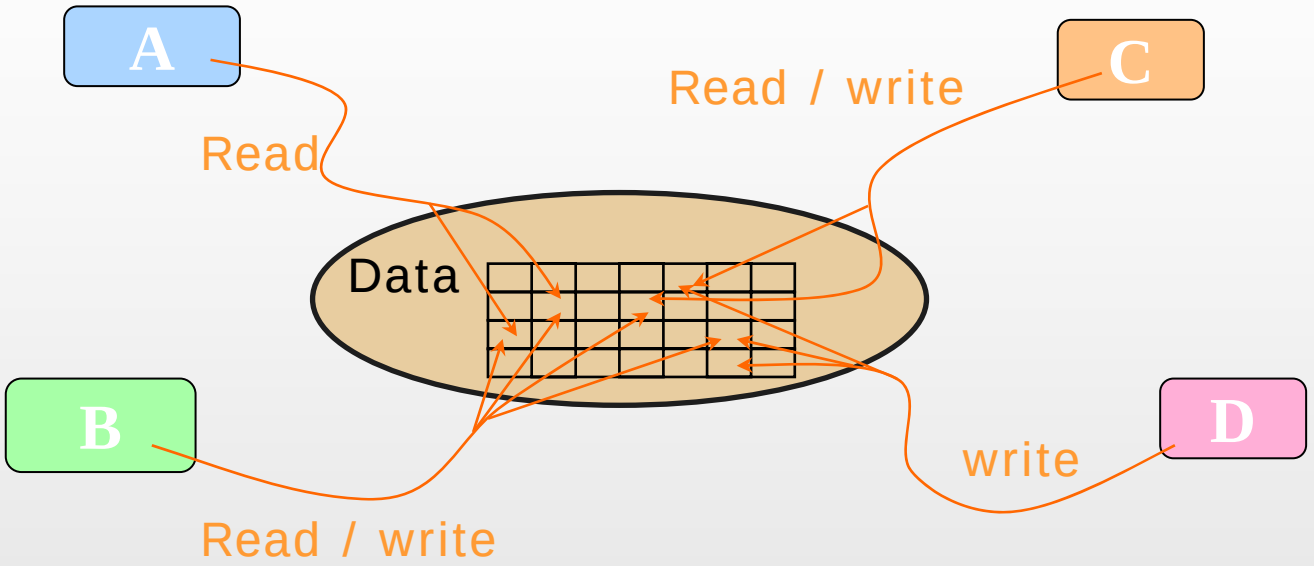
- World climate behavior
  - Energy transport: Equator ⇨ Pole
- Parameterization design
  - Independent and simultaneous simulations
- Code coupling
  - ARPEGE v4.5 (atmospheric simulation)
  - OPA v9 +LIM (ocean simulation)
  - OASIS v3 (code coupling)



- Master-Worker + Data Sharing
- Platform target
  - Grid



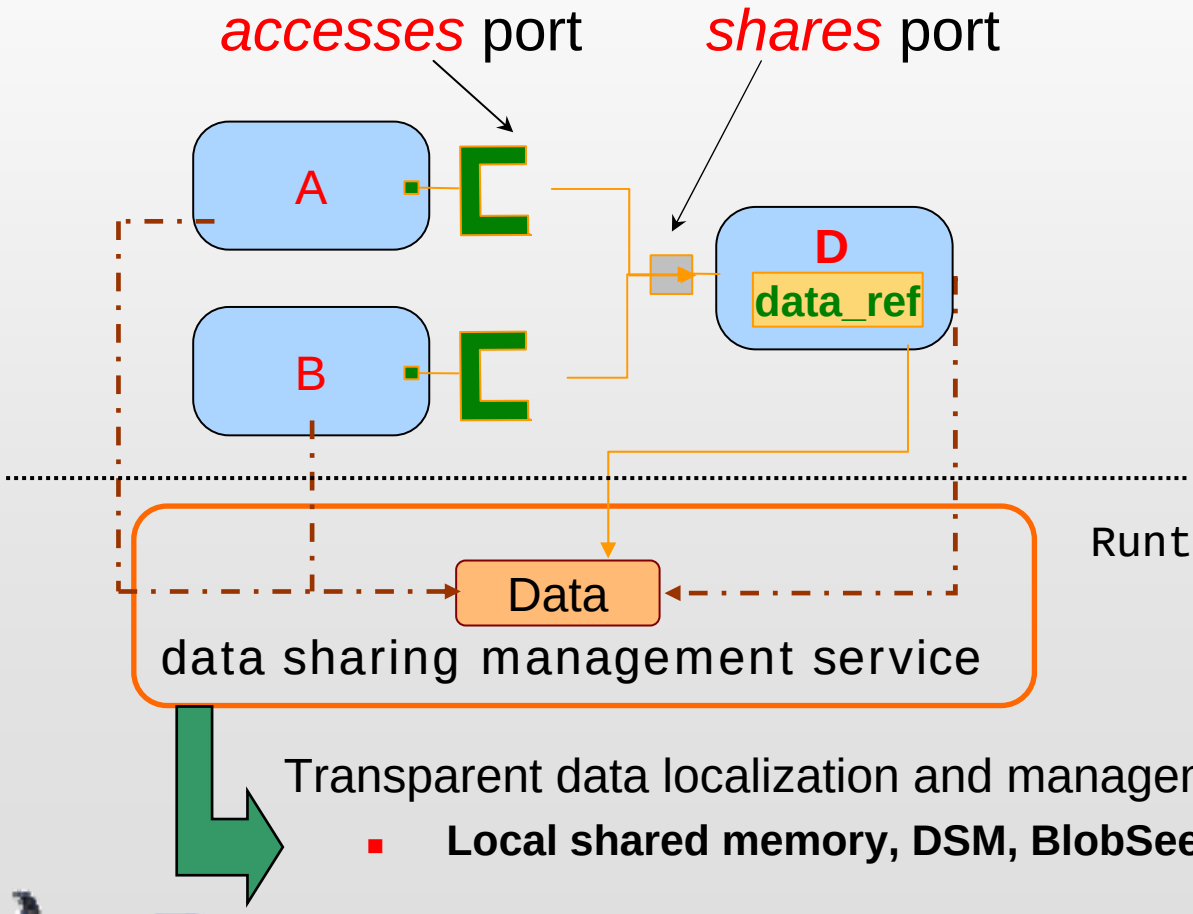
Data Sharing Overview



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

Data Shared Port Model

- Principle: rely on a data sharing management service



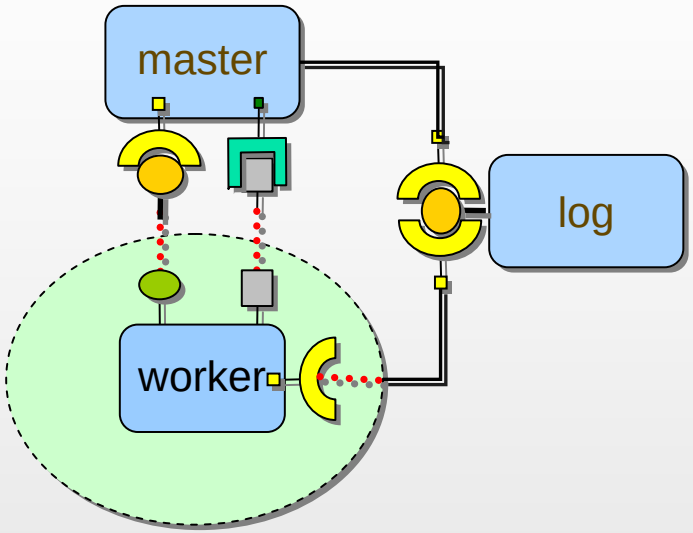

```
component D {
  shares array<float> u;
};

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

```
interface data_handling
 {
    float* get_pointer();
    long get_size();
    void  acquire();
    void  acquire_read();
    void  release();
}
```

Transparent data localization and management

- **Local shared memory, DSM, BlobSeer, …**

Climatology Application
Composing Middleware: The Model

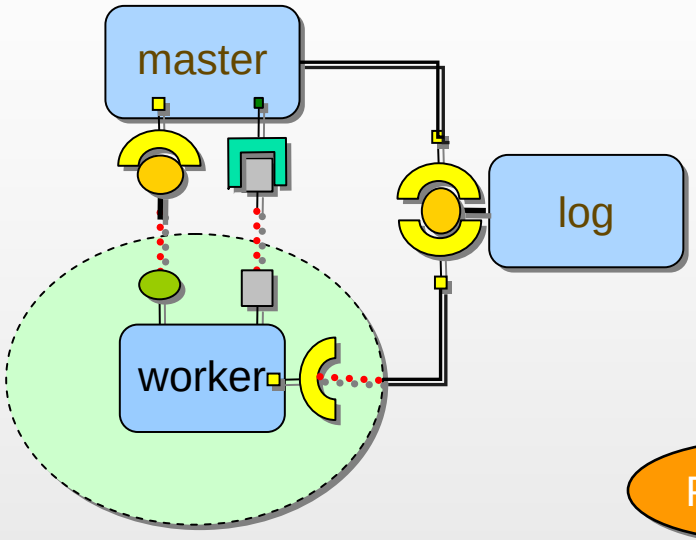


Primitive Worker Component:

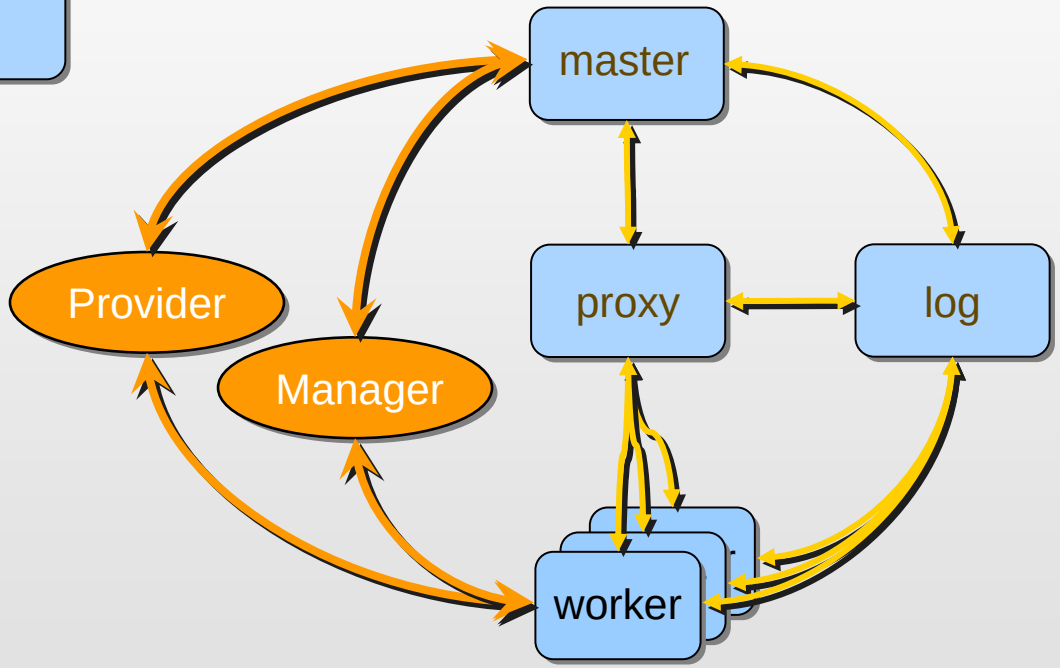
- MPI-based code coupling (OASIS)

Description in ULCM
(~50 lines)

Climatology Application
Composing Middleware: An Execution Model



Description in ULCM
(~50 lines)



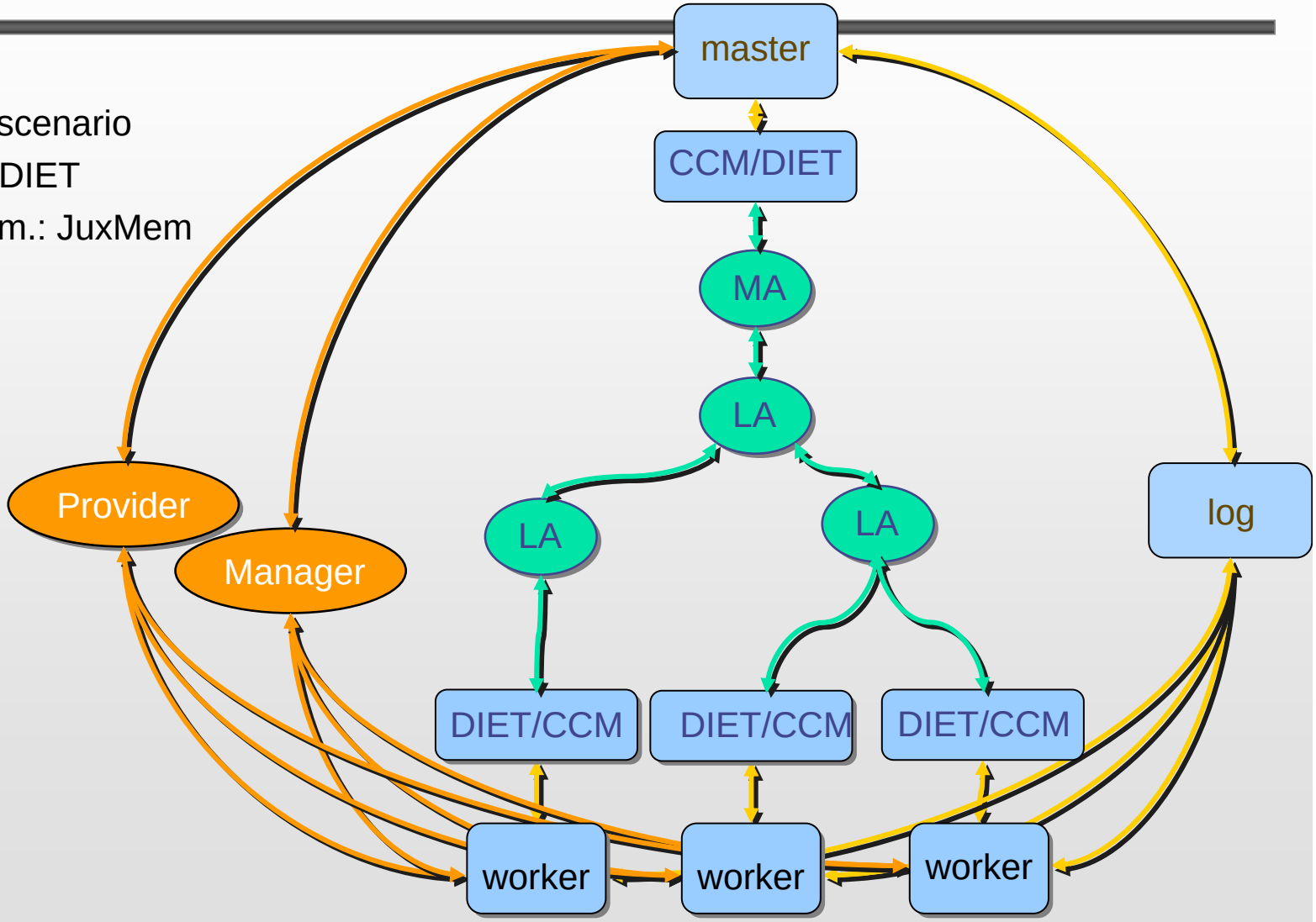
Simple scenario

- Collection: CCM Proxy based pattern
- Shared Mem.: JuxMem

Climatology Application
Composing Middleware: Another Execution Model

More complex scenario

- Collection: DIET
- Shared Mem.: JuxMem



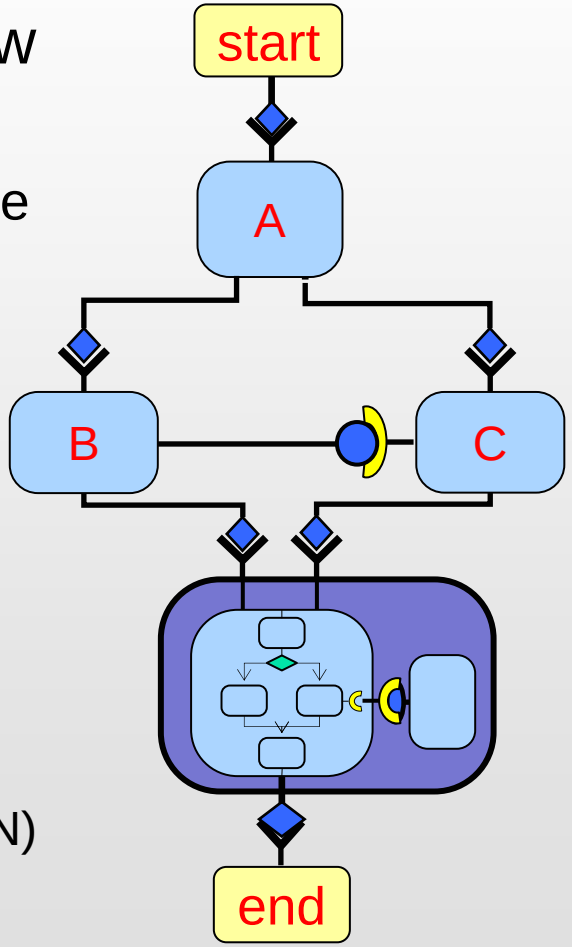
Spatial and Temporal Composition

Workflow Models vs. “Classical” Component Models

	Assembly	Simplicity	Code coupling	Resources usage
Workflow models	+	+	-	+
Component models	+	+	+	-

Principle of STCM: Spatio-Temporal Component Model

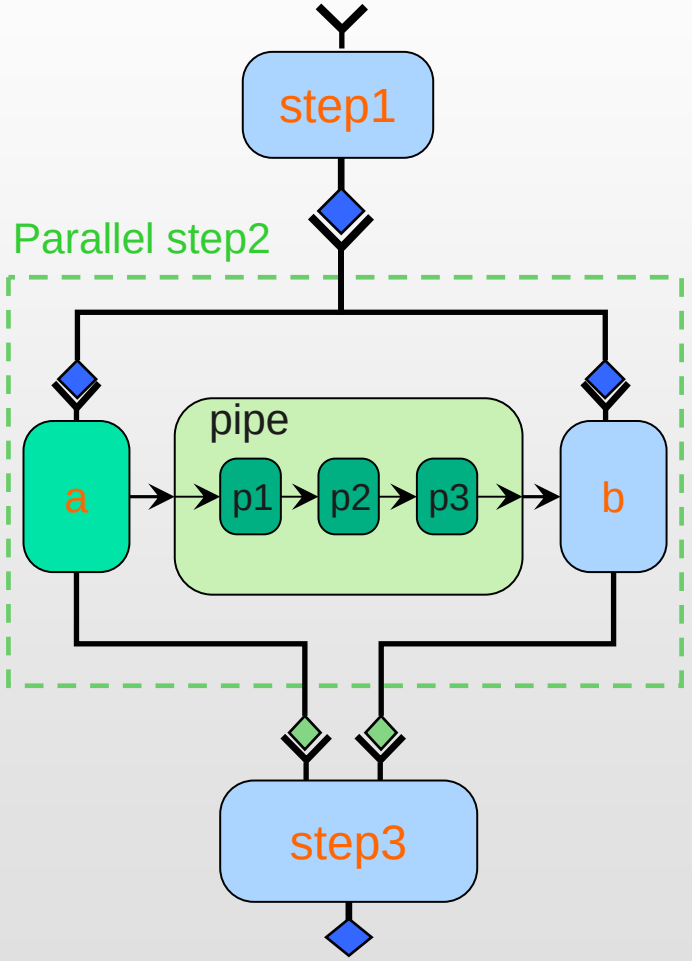
- Combination of component and workflow models
  - Spatial and temporal dimensions at the same level of assemblies
- Component-task
  - Input and output ports (temporal)
  - Spatial ports
  - Task
- Assembly model
  - Adaptation of a workflow language
    - Abstract Grid Workflow Language (ASKALON)



And Algorithmic Skeletons?

Overview of STKM: Spatio-Temporal Skeleton Component Model

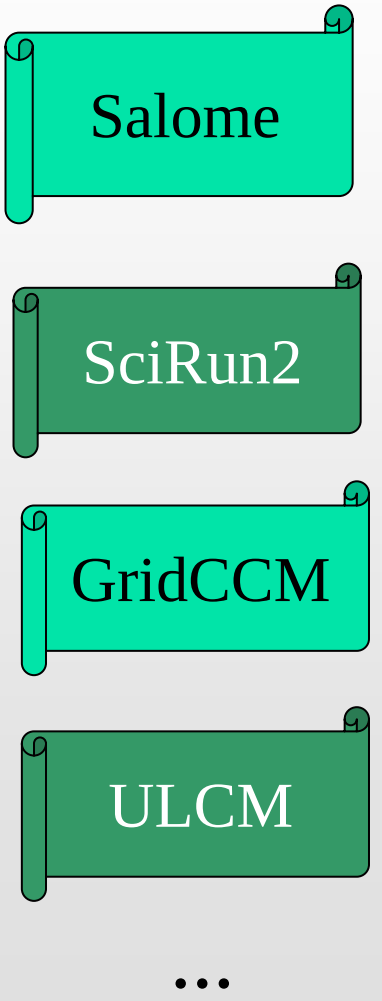
- Assembly model
  - STCM assembly + skeleton constructs
  - An STKM skeleton is a composite with a predefined behavior
  - Parameterization
    - Wrapping components
  - Usage in spatial and temporal dimension
    - Port cardinality principle (temporal dimension)



So?

A Rich Set of Composition Operators

- At port level
  - Data shared port
  - Collective communication
- At component level
  - SPMD paradigm (MxN communication)
- At assembly level
  - Collective operations
  - Master-worker paradigm
- Assembly kind
  - Algorithmic Skeletons
  - Spatial and temporal composition
  - …

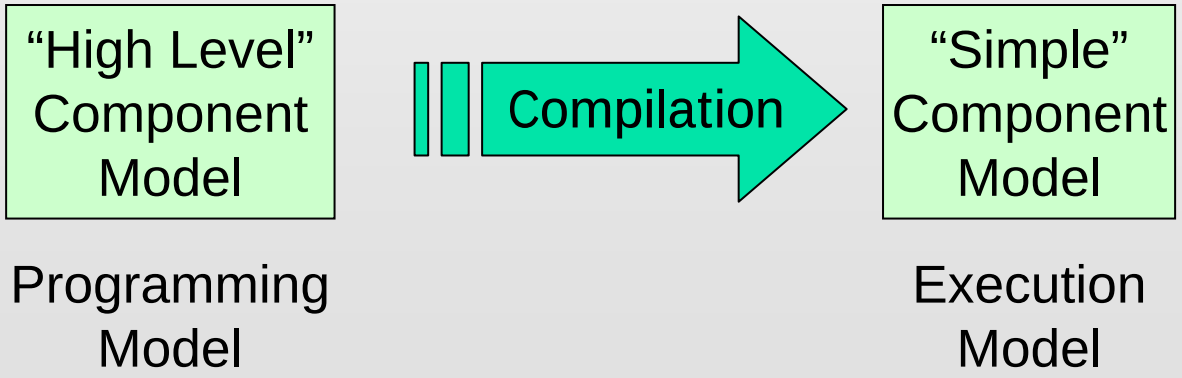


From Programming Model to Efficient Execution Model

How to implement all these composition operators?

Towards ADL Compilation

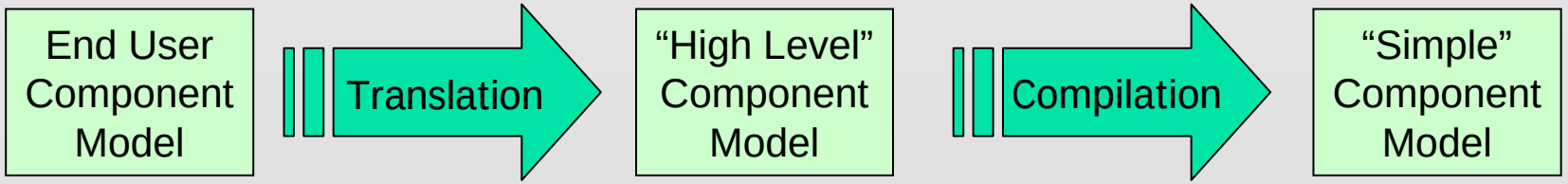
- All “advanced” component construct implementations are based on some transformations to a “low/execution” level component model
- Compilation enables to adapt abstraction to the actual resources
  - Transparent adaptation if the compilation is on the fly



Dedicated or not Implementation?

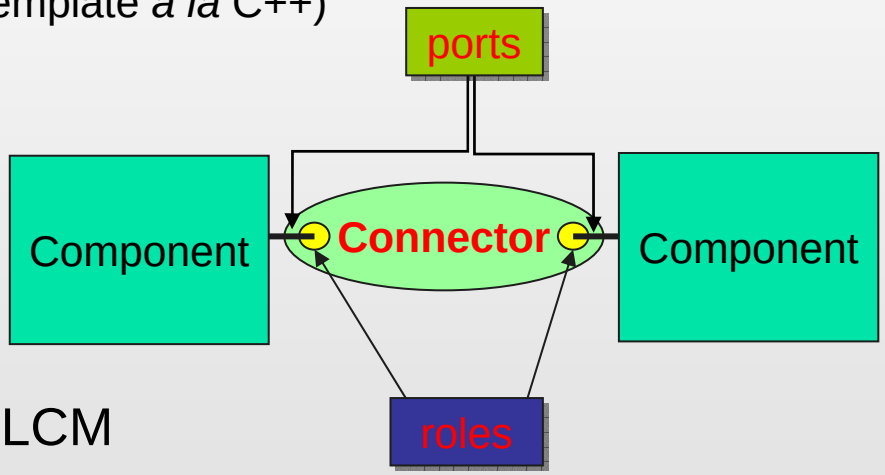
- Dedicated implementation (aka “classical” way)
  - Define a model with a fixed number of composition operators
  - Example:
    - ULCM: Spatial & Temporal, Master-Worker, Data Sharing
- ➢ How to add new composition operators?

- Towards a non-dedicated implementation
  - Define an intermediate model for expert to let them define composition operators



HLCM: High Level Component Model

- Major concepts
  - Hierarchical model
  - Generic model
    - Support meta-programming (template *à la* C++)
  - Connector based
    - Primitive and composite
  - Currently static



- HLCMi: an implementation of HLCM
  - Model-transformation based
  - Already implemented connectors
    - Use/Provide, Shared Data, Collective Communications, MxN, some Skeletons

Conclusion

Conclusion

- Software component is a promising technology for handling code & resource complexity
- Component model as a *programming* model
  - A lot of composition operators has been already defined & prototyped
  - Re-use existing specialized middleware
  - Good feedback from users
- Open issues
  - Other operators? Finer grain?
    - MapReduce, Domain decomposition, AMR, …
  - HLCM expressiveness and performance
    - Temporal composition seems possible
  - Automatic component/connector selection & configuration
    - Need to interact with resources (cf French ANR COOP project)
  - …

Acknowledgement

- Thierry Priol (INRIA)
- Marco Danelutto (U. Pisa)
- Eric Maisonnave (CERFACS)
- Jocelyne Erhel (INRIA)
- Andre Ribes (EDF R&D)
- Hinde Bouziane (INRIA)
- Julien Bigot (INRIA)
- …